BaroKEY

nurit

BaroKEY 가이드(C)

문 서 번 호	BARO-KEY-004
문 서 버 전	1.0
작 성 일 자	2019.02.22
작 성 자	이 종 일

승인 내역

구분	소속/직책	성명	일자	서명
	누리아이티/기술이사	이종일	2019.02.22	

개정 이력

개정 번호	개정 범위	개정 내역	개정자	개정 일자
1.0	전체	최초 작성	이종일	2019.02.22

목차

1. 일회용 인증키(One Time Authentication key)

1.1 일회용 인증키 개요

전자거래 중에 교환되는 데이터가 변경되지 않고 허가되지 않은 제3자에 의해 간섭받지 않도록하기 위해 사용되는 디지털(전자) 키로 그것은 인증 및 보안 트랜잭션의 고유 한 정보(양, 날짜, 시간)를 기반으로하는 디지털(전자) 코드 세트로 구성된다. 일명 디지털 키 또는 전자 키라고도 불린다.

고정된 비밀번호 대신 무작위로 생성한 **일회성 인증키**로 사용자를 인증하는 방법으로, 보안을 강화하려고 도입한 시스템이다. 로그인 할 때마다 **일회성 인증키**를 생성한다. 동일한 **일회성 인증키**가 사용되지 않아 보안을 강화할 수 있다. 주로 전자 거래에서 사용된다.

사용자는 "**일회용 인증키**"를 생성하는 하드웨어인 **일회성 인증키**의 생성기(Authentication key Token)를 이용한다. 별도 **일회성 인증키**의 생성기를 소지해야 하는 불편함으로 전자 거래에만 주로 사용된다. **일회성 인증키**의 생성기는 버튼을 누르면 8자리의 인증키가 생성되는 방식이다.

매 1분마다 자동으로 서로 다른 8자리의 **일회성 인증키**가 나오는 시간 동기 방식과 키패드에 4자리 비밀번호를 입력하면 8자리 **일회성 인증키**를 보여 주는 방식 등이 있다. 형태로는 소형 단말기 형태 토큰형과 카드형이 사용된다. 최근에는 휴대폰 범용 가입자 식별모듈(USIM)을 기반으로 하는 모바일 인증키(Mobile Authentication key)의 도입이 늘고 있다.

1.2 인증키 생성 및 인식 방식

1. S/KEY 방식

벨 통신 연구소에서 개발한 인증키 생성 방식으로, 유닉스 계열 운영 체제에서 인증에 사용되고 있다. 그 생성 알고리즘은 아래와 같다.

① 클라이언트에서 정한 임의의 비밀키를 서버로 전송한다.
② 클라이언트로부터 받은 비밀키를 첫 값으로 사용하여, 해시 체인 방식으로, 이전 결과 값에 대한 해시 값을 구하는 작업을 n번 반복한다.
③ 그렇게 생성된 n개의 인증키를 서버에 저장한다.

또한 이를 통해, 클라이언트에서 i번째로 서버에 인증을 요구할 때의 인증 방식은 아래와 같다.

① 클라이언트에서 정한 인증키에 해시 함수를 n-i번 중첩 적용하여 서버로 전송한다.
② 서버에서는 클라이언트로부터 받은 값에 해시 함수를 한 번 적용하여, 그 결과가 서버에 저장된 n-i+1번째 인증키와 일치하는지 검사한다.
③ 일치하면 인증에 성공한 것으로, 카운트를 1 증가시킨다.

해시 체인에 기반하고 있는 이 알고리즘은, 해시 함수의 역연산을 하기 어렵다는 점에 착안하여 만들어졌다.

해킹에 의해 클라이언트에서 서버로 전송되는 값이 노출된다 해도, 그 값은 일회용이므로 다시 사용될 수 없다. 또한 그 다음 인증을 위해서는 그 전에 전송된 값보다 해시 함수가 한 번 적게 적용된 값을 사용해야 하는데, 그 값을 유추하는 것 또한 매우 어렵다.

다만 생성했던 인증키를 모두 소진하면 다시 새로 설정을 해야 하며, 서버에 저장돼 있는 인증키 목록이 유출될 경우 보안에 대단히 취약해진다는 단점이 있다.

2. 시간 동기화 방식

인증키를 생성하기 위해 사용하는 입력 값으로 시각을 사용하는 방식이다. 클라이언트는 현재 시각을 입력값으로 인증키를 생성해 서버로 전송하고, 서버 역시 같은 방식으로 인증키를 생성하여 클라이언트가 전송한 값의 유효성을 검사한다.

임의의 입력값이 필요하지 않다는 점에서 사용하기 간편하고, 클라이언트가 서버와 통신해야 하는 횟수가 비교적 적다. 또 서버에서 클라이언트에 입력값을 보내는 방식이 아니므로, 여타 인증키 생성 방식에 비해 피싱에 안전하다. 한편 클라이언트에서 시각 정보를 이용해 인증키를 생성하므로, 스마트폰 등의 모바일 기기도 클라이언트로 사용되기 적합하다는 점 역시 비용 절감 측면에서 장점이다.

하지만 클라이언트와 서버의 시간 동기화가 정확하지 않으면 인증에 실패하게 된다는 단점이 있으며, 이를 보완하기 위해 일반적으로 1~2 분 정도를 인증키 생성 간격으로 둔다.

미국 RSA사에서 만든 '시큐어 ID'가 이 방식을 사용한다.

3. 챌린지.응답 방식

서버에서 난수 생성 등을 통해 임의의 수를 생성하고 클라이언트에 그 값을 전송하면, 클라이언트가 그 값으로 인증키를 생성해 응답한 값으로 인증하는 방식이다.

입력값이 매번 임의의 값이 된다는 측면에서는 안전성을 갖추고 있으나, 네트워크 모니터링에 의해 전송되는 값들이 노출될 경우 매우 취약해진다는 단점이 있다. 또 서버와 클라이언트 사이의 통신 횟수도 비교적 많이 요구된다.

4. 이벤트 동기화 방식

서버와 클라이언트가 카운트 값을 동일하게 증가시켜 가며, 해당 카운트 값을 입력값으로 인증키를 생성해 인증하는 방식이다.

다만 클라이언트에서 인증키를 생성하기만 하고 인증에 사용하지 않으면, 서버와 클라이언트의 카운트 값이 불일치하게 된다는 문제점이 있다. 이러한 문제를 보완하기 위해 어느 정도 오차 범위 내에서는 인증을 허용하는 방법이나, 카운트가 어긋났다고 판단될 경우 연속된 인증키를 받아 유효성을 판별하는 방법 등이 사용된다.

1.3 인증키 전달 방식

인증키는 사용자 인증에 사용되므로, 기본적으로 서버와 클라이언트 사이에 통신 및 전달 수단이 필요하다. 이러한 수단으로는, 아래와 같은 것들이 있다.

1. 인증키 토큰

인증키 토큰이라 불리는 별도의 하드웨어를 클라이언트로 사용하는 방식이다. 기기 자체에서 해킹이 이루어지기는 힘들지만, 토큰을 구입해야 하므로 추가 비용이 필요하며 휴대하기에 불편하다는 단점이 있다.

2. 카드형 인증키 토큰

기존의 토큰의 불편한 휴대성을 개선하기 위해, 얇은 두께로 휴대하기에 편리한 카드형 토큰도 출시된 바 있다. 일반 카드에 비해 별로 두껍지 않아 지갑에도 휴대가 가능하지만, 처리 속도가 느려 인증키 생성에 시간이 더 걸리고 수명이 짧으며 가격도 비싸다는 단점이 있다.

3. 스마트폰 앱

별도의 하드웨어 장비를 필요로 하지 않아서 추가 비용 없이도 인증키 서비스를 이용할 수 있는 방식이다. 해당 스마트폰에 맞게 제공되는 앱을 설치함으로써 이용할 수 있다. 물론 서버 측에서 이 방식을 지원하지 않으면 이용할 수 없다는 단점이 있다.

4. SMS

SMS로 인증키를 전달하는 방식이다. 스마트폰이 아닌 어떤 종류의 휴대전화만 있어도 이용 가능하다는 장점이 있지만, SMS 자체의 해킹 위험성에 의해 현재는 거의 사용되지 않고 있다.

1.4 메시지 인증코드

1. 메시지 인증코드란?

메시지인증코드(Message Authentication Code)의 약어로서

■ 이 기술은 메시지에 붙여지는 작은 데이터 블록을 생성하기 위해 비밀키를 이용하는 방법이다.
■ 이 기술을 이용하면 전송되는 메시지의 무결성을 확인하여, 메시지에 대한 인증을 할 수 있다.
■ 메시지 인증 코드는, 임의 길이의 메시지와 송신자와 수신자가 공유하는 키라는 2개의 입력을 기초로 해서, 고정 비트 길이의 출력을 계산하는 함수이다.
■ 이 함수의 출력을 MAC 값이라 부른다.

MAC는 메시지의 인증에 쓰이는 작은 크기의 정보이다. 이 MAC을 이용하여, 메시지의 무결성과 신뢰성을 보장하는데 사용한다.

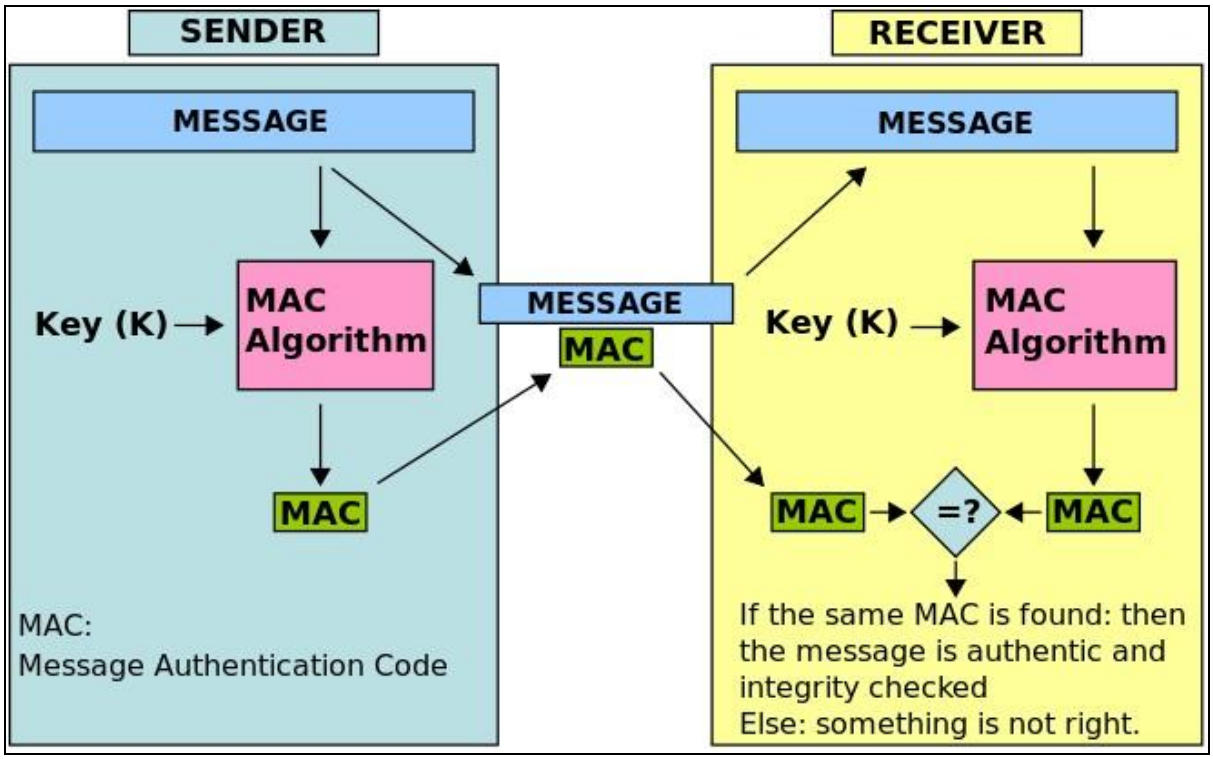
MAC의 알고리즘은 인증을 위한 Secret Key와의 임의 길이의 메시지를 입력 받아 MAC을 출력하는 Keyed Hash Function을 사용한다.

MAC은 Cryptographic Hash Function과 같은 특성을 가진다. 그 메인 속성은 다음과 같다.

① Hash value로 계산하기 쉽다.
② 생성된 Hash를 통해서 Message를 Generate 하는 것이 불가능하다.
③ Hash를 수정하지 않고, Message를 수정하는 것이 불가능하다.
④ 다른 두 Message가 동일한 Hash를 보내는 것이 불가능하다.

사용되는 Hash 알고리즘은 MD5, SHA-1, SHA-2 등 일반적인 암호화 알고리즘을 그대로 사용할 수 있으며, 사용된 알고리즘에 따라서 고정 길이의 Hash value가 생성된다.

MAC을 간단하게 도식화 하면 다음과 같다.



MAC에 사용되는 Key에 따라서 CMAC, HMAC, UMAC, VMAC 등으로 나눈다.

① CMAC
암호 기반 메시지 인증 방식으로 Cipher-based MAC이다.
AES와 triple-DES를 이용하는 방법으로 많이 사용된다.

② UMAC
Universal Hashing 기반으로 메시지 인증코드를 사용하는 방법이다.
Universal Hashing는 다음의 특성을 가지는 Hash 함수 F를 선택하기 위한 확률적 알고리즘이다.

F(x) = F(y)

③ VMAC
블럭 함호 기반 메시지 인증(block cipher-based message authentication code) 방식으로 보편적 해시 알고리즘을 사용한다.

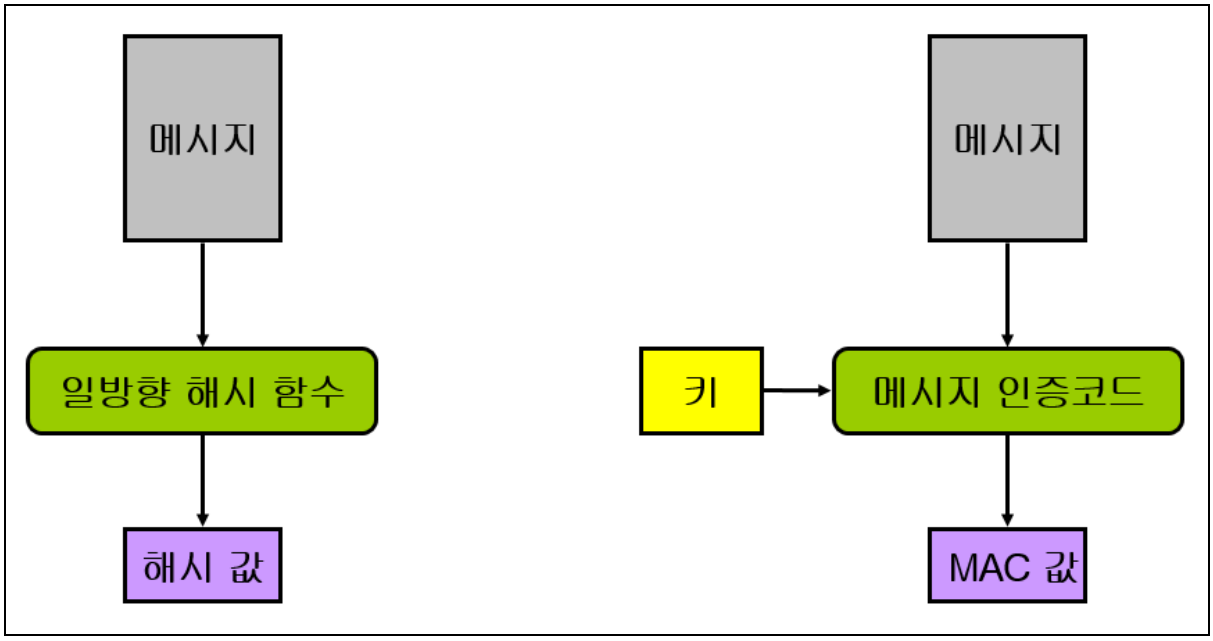
이런 것을 사용하는 이유는 다음과 같다.

① 짧고, 고정 길이 이다. ② 중복을 방지할 수 있다. ③ 메시지 구조를 숨길 수 있다. ④ 그러면서도 메시지에 대한 유효성 및 인증이 가능하다.

암호화 알고리즘과 동일한 방법을 사용하므로, 알고리즘에 따라서 속도가 많이 달라진다. 속도는 MD%가 빨라서 많이 사용했으나, 암호화 결함이 발견되어 사용이 줄어 들었다.

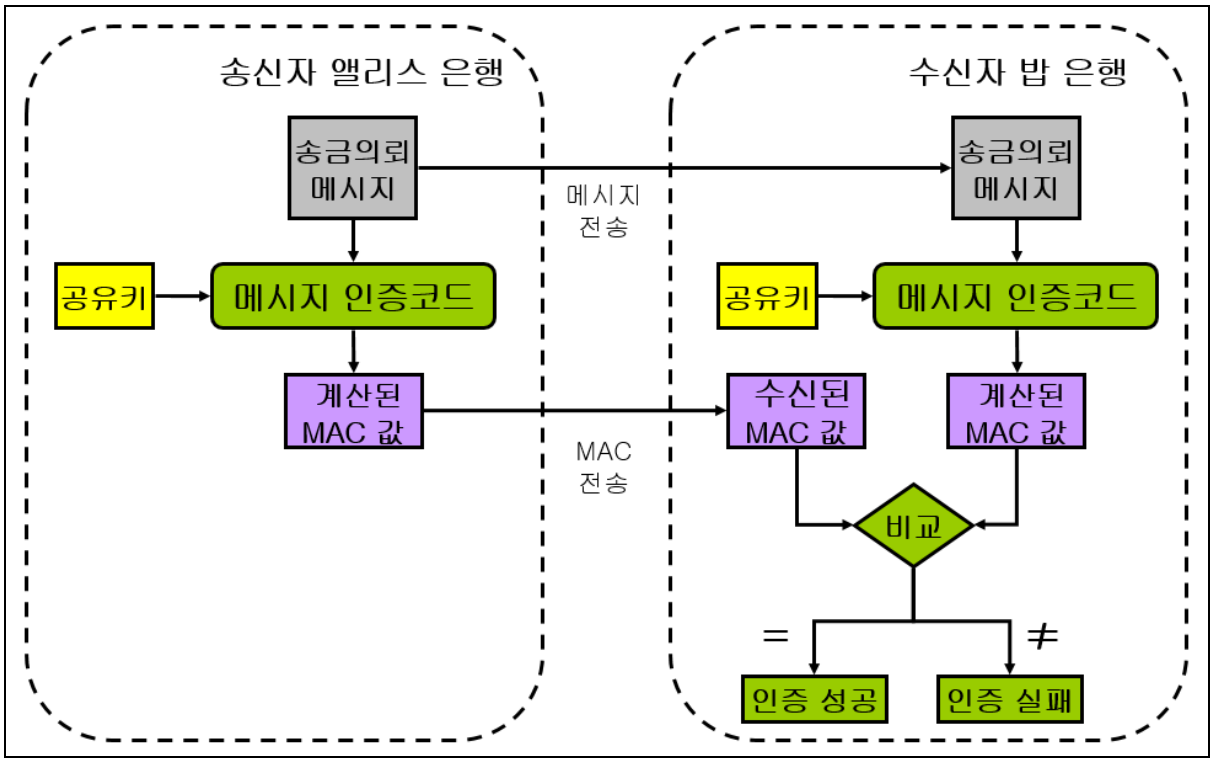
보완 관련 용도는 SHA-256을 권장하지만, 아직까지는 한국에서는 SHA-1을 많이 사용한다.

1) 일방향 함수와 MAC의 차이점



■ 일방향 해시 함수로 해시 값을 계산할 때는 키를 사용하지 않았다.
■ 메시지 인증 코드에서는 송신자와 수신자가 공유하는 키를 사용한다.

2) MAC의 이용순서



MAC은 Cryptographic Hash Function과 같은 특성을 가진다. 그 메인 속성은 다음과 같다.

① 주어진 Message를 Hash값으로 계산하기는 쉽다.
② 주어진 Hash값으로 Message를 계산하는 것은 거의 불가능하다.
③ 아주 약간의 차이를 가진 두 Message가 동일한 Hash값을 갖는 일은 거의 불가능하다.

2. HMAC(Hash-based Message Authentication Code)란?

해시 함수(hash function)는 임의의 길이를 갖는 메시지를 입력 받아 고정된 길이의 해쉬값을 출력하는 함수이다. 암호 알고리즘에는 키가 사용되지만, 해쉬 함수는 키를 사용하지 않으므로 같은 입력에 대해서는 항상 같은 출력이 나오게 된다. 이러한 함수를 사용하는 목적은 입력 메시지에 대한 변경할 수 없는 증거값을 뽑아냄으로서 메시지의 오류나 변조를 탐지할 수 있는 무결성을 제공하는 목적으로 주로 사용된다.

해쉬 함수는 전자 서명과 함께 사용되어 효율적인 서명 생성을 가능하게 한다. 긴 메시지에 대해 서명을 하는 경우, 전체 메시지에 대해 직접 서명을 하는 것이 아니고 짧은 해쉬값을 계산해 이것에 대해 서명을 하게 된다. 공개케 연산은 많은 계산량을 필요로 하기 때문에 전체 미시지를 공개키 길이의 블록 단위로 나누어 모든 블록에 대해 서명을 하는 것은 매우 비효율적이다. 그러므로 먼저 메시지를 입력 받아 짧은 해쉬값을 계산하고, 이것에 대해 한 번의 서명 연산을 하는 것이다. 이 계산값은 원래의 메시지에 대한 서명으로 인정된다.

해쉬값에 대한 서명이 원 메시지에 대한 서명으로 인정되기 위해서는 같은 해쉬값을 갖는 또 다른 메시지를 찾아내기가 계산적으로 어려워야 한다. 해쉬 함수는 임의의 길이의 입력으로부터 짧은 길이의 해쉬값을 출력하므로 입력은 서로 다르지만 같은 출력을 내는 충돌이 반드시 존재한다. 만일 같은 해쉬값을 갖는 다른 메시지를 찾아내기가 쉽다면, 서명자는 자신의 서명에 대해 다른 메시지에 대한 서명이라고 우길 수 있을 것이기 때문이다. 이것이 가능하다면, 전자 서명에 대한 신뢰가 불가능하고 전자 거래에 사용할 수 없게 될 것이다. 그러므로 안전한 해쉬 함수로 사용될 수 있기 위해서는 충돌을 찾아내기 어렵다는 특성을 가져야 한다.

한편, 해쉬 함수는 전자 서명에 사용된다고 했는데, 이것은 서명자가 특정 문서에 자신의 개인키를 이용하여 연산함으로서 데이터의 무결성과 서명자의 인증성을 함께 제공하는 방식이다. 메시지 전체에 대해 직접 서명하는 것은 공개키 연산을 모든 메시지 블록마다 반복해야 하기 때문에 매우 비효율적이다. 따라서 메시지에 대한 해쉬값을 계산한 후, 이것에 대해 서명함으로서 매우 효율적으로 전자 서명을 생성할 수 있다. 서명자는 메시지 자체가 아니라 해쉬값에 대해 서명을 하였지만 같은 해쉬값을 갖는 다른 메시지를 찾아내는 것이 어렵기 때문에, 이 서명은 메시지에 대한 서명이라고 인정된다.

송신자의 신분에 대한 인증이 필요 없고, 데이터가 통신 중 변조되지 않았다는 무결성만이필요한 경우에는 해쉬 함수를 메시지인증코드(MAC, Message Authentication Code)라는 형태로 사용할 수 있다. 송신자와 수신자가 비밀키를 공유하고 있는 경우, 송신자는 메시지와 공유된 비밀키를 입력으로 하여 해쉬값을 계산하면 메시지인증코드가 된다. 메시지와 함께 메시지인증코드를 함께 보내면 수신자는 메시지가 통신 도중 변조되지 않았다는 확신을 가질 수 있다.

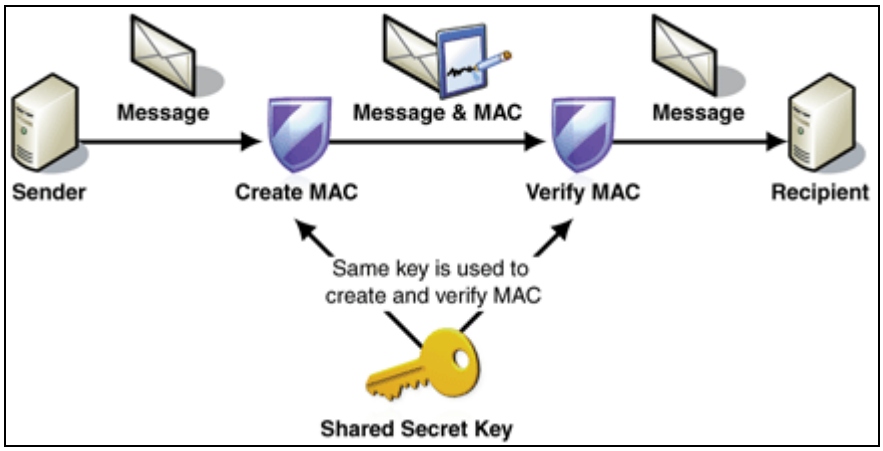
대표적인 해쉬 함수는 다음과 같다.

최조의 알고리즘은 1993년 미국 국가 안전 보장국(NSA)이 설계하였으며, 미국 표준 기술 연구소(NIST)에 의해 SHS(Secure Hash Standard, FIPS PUB 180)으로 출판되었으며, 다른 함수들과 구별하기 위해 보통 SHA-0으로 불린다. 2년 후 SHA-0의 압축 함수에 비트 회전 연산을 하나 추가한 SHA-1(FIPS PUB 180-1)이 발표되었으며, 그 후에 4종류의 변형, 즉 SHA-224, SHA-256, SHA-384, SHA-512(FIPS PUB 180-2)가 추가로 발표되었다. 이들을 통칭해서 SHA-2라고 하기도 한다. SHA-1은 SHA 함수들 중 가장 많이 쓰이며, TLS, SSL, PGP, SSH, IPSec 등 많은 보안 프로토콜과 프로그램에서 사용되고 있다. 하지만 최근 SHA-0과 SHA-1에 대한 분석 결과가 발표됨에 따라 SHA-2를 사용할 것이 권장되고 있다.

국내에서 개발된 대표적인 해쉬 함수인 HAS-160[28]은 2000년 12월 국내 해쉬 함수 표준(TTAS.KO-12.0011/R1)으로 채택되었다. HAS-160의 설계 원리는 SHA-1의 설계 사상이 유사하지만, 메시지 확장 과정

이 기존의 MD계열 해쉬 함수와는 차이가 있어, 최근 제안된 다양한 해쉬 함수 분석 기법에 대하여 아직까지는 안전성을 가지고 있다.

한마디로 HMAC는 Key를 조합하여 Hash 함수를 구하는 방식으로 송신자와 수신자만이 공유하고 있는 Key와 메시지를 혼합하여 Hash 값을 만드는 방식이며, 채널을 통해 보낸 메시지가 훼손되었는지 여부를 확인하는데 사용할 수 있으며, MAC 특성상 역산이 불가능하므로, 수신된 메시지와 Hash 값을 다시 계산하여, 계산된 HMAC과 전송된 HMAC이 일치하는지를 확인하는 방식이다.



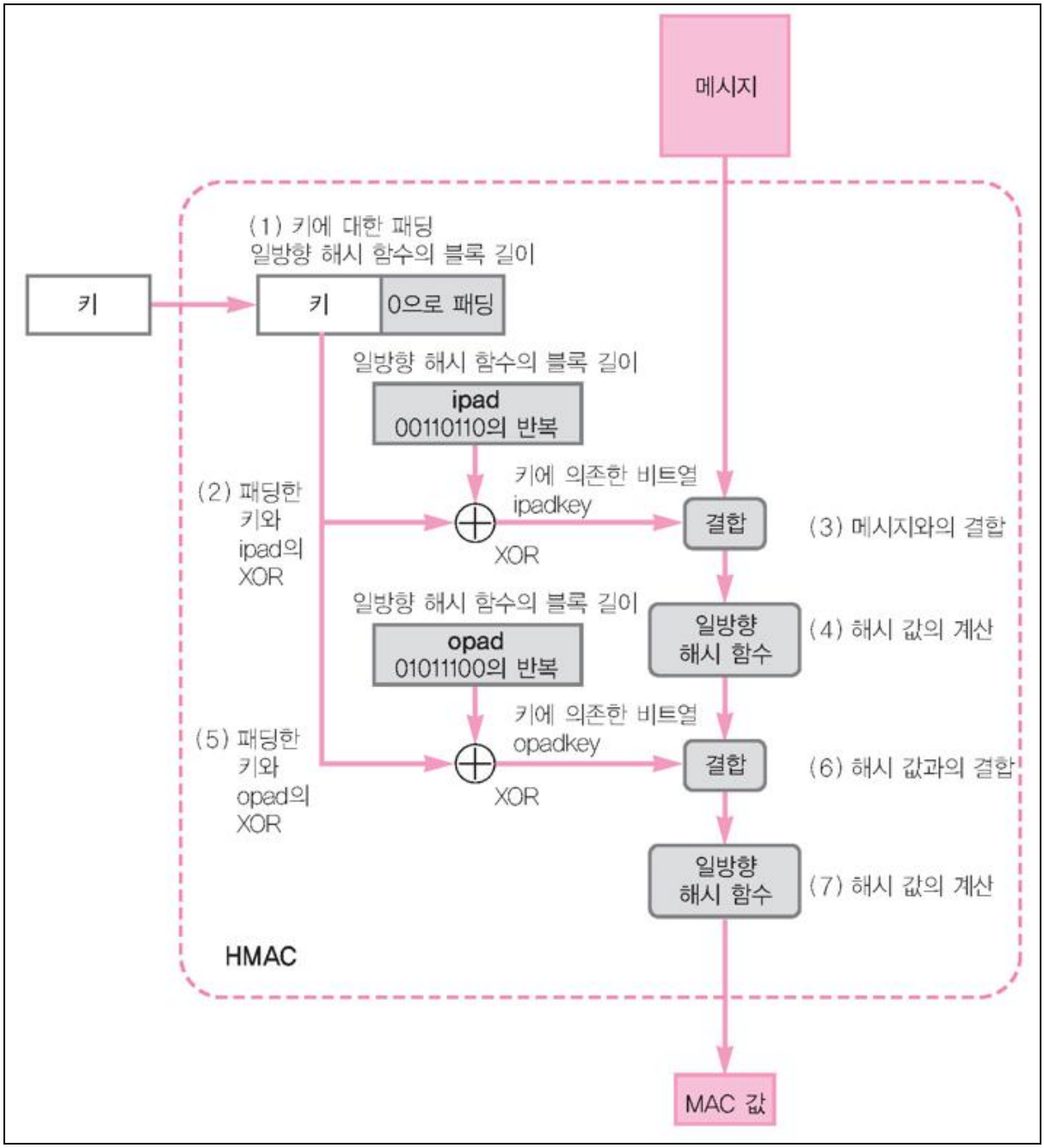
① HMAC은 일방향 해시 함수를 이용해서 메시지 인증 코드를 구성하는 방법인데, HMAC에서는 사용하는 일방향 해시 함수를 단 한 종류로 정해 놓고 있는 것은 아니다.

② 강한 일방향 해시 함수라면 뭐든지 HMAC에 이용할 수 있다.

③ 장래 새로운 일방향 해시 함수가 고안된다면 그것을 사용할 수도 있다.

④ 이와 같은 형태로 만들어진 알고리즘을 모듈형 알고리즘이라고 한다.

HMAC 의 순서는 다음과 같이 단방향 해시 함수를 사용한 메시지 인증코드의 예



대칭블록암호에 기반을 둔 MAC 방식으로 알고리즘에 따라서 HMAC value size가 달라진다.

```
HMAC_MD5    ("", "") = 0x74e6f7298a9c2d168935f58c001bad88
HMAC_SHA1   ("", "") = 0xfbdb1d1b18aa6c08324b7d64b71fb76370690e1d
HMAC_SHA256("", "") = 0xb613679a0814d9ec772f95d778c35fc5ff1697c493715653c6c712144292c5ad
```

암호화 알고리즘과 동일한 방법을 사용하므로, 알고리즘에 따라서 속도가 많이 달라진다. 속도는 MD5가 빨라서 많이 사용했으나, 암호화 결함이 발견되어 사용이 줄어들었다. 보안관련 용도는 SHA-256을 권장하지만, 아직까지 한국에서는 SHA-1을 많이 사용한다.

2. BaroKEY

BaroKEY 솔루션은 ERP, 전자결재, 그룹웨어 등 기업 내의 어플리케이션 로그인 시 비밀번호를 대체 또는 로그인-ID, 비밀번호 이외에 추가 인증(2차 인증)하는 소프트웨어 방식(어플리케이션 레벨)의 인증 솔루션으로, 일명 소프트 인증키, 2세대 인증키, 일회용 인증키로 불린다.

2.1 본인인증 방식

개인정보가 포함된 본인인증 방식은 크게 IP 접속제한, 공인인증서, 일회용 인증키, 생체인식 방식 등 4가지로 구분 할 수 있다.

IP 접속 제한	공인 인증서	일회용 인증키	생체인식
IP 접속 제한 방식은 정보 자산(Windows, MacOS, Linux, Unix,Database, 네트워크 장비, 보안장비, 저장장치 등)에 고정된 IP가 필요하기 때문에 IP가 변하는 유동 IP를 사용한다면 접속 제한에 의미가 없음.	공인 인증서 방식은 솔루션 비용이 비싸며, 공인 인증서 인증 모듈의 불편함 때문에 사용하기가 힘들고, 공인 인증서 의무화가 폐지되어 대체할 대안이 필요함.	일회용 인증키 방식은 누구나 사용하기 쉽고, 시간과 장소의 제약을 받지 않으며, 본인이 소유하고 있는 스마트 폰에서 인증키를 생성하기 때문에 간편한 인증 방법인 동시에 한번 사용된 인증키는 재사용할 수 없으며, 인증키 유추가 어려워 다양한 해킹 공격에도 강력한 보안성을 제공함.	요즘 각광 받고 있는 생체인식 방식은 개인의 신체 특성을 활용하여 개인별로 유일하기 때문에 강력한 보안성을 제공하지만 솔루션 비용이 비싸며, 비밀번호는 해킹을 당할 경우 변경하면 그만이지만, 생체정보를 해킹 당할 경우에는 변경이 거의 불가능하다. 따라서 최악의 경우 영구적인 피해로 이어질 수도 있음.

2.2 인증키 구분 및 특징

일회용 인증키는 인증서버가 필요한 Hard 방식의 1st 인증키(1세대 인증키)와 별도의 인증서버가 존재하지 않아 관리가 필요 없고, 손쉽게 적용할 수 있는 소프트웨어 방식(모듈 호출)의 2nd 인증키(2세대 인증키)로 구분한다.

1st 인증키(Hard 인증키)	2nd 인증키(Soft 인증키)
❖ 서버 인증 방식(SHA-I)	❖ 소프트웨어 인증 방식(SHA-II, SHA-III)
❖ 토큰, 카드 위주(인증키 생성기)	❖ 스마트폰 위주(인증키 생성기)
❖ 개인별 HMac Key 발급 및 관리	❖ 개인별 HMac Key 발급 및 관리하지 않음
❖ 정적 HMac Key 방식	❖ 동적 HMac Key 방식
❖ 일괄 인증키 생성주기(30, 60초) 적용	❖ 개별 인증키 생성주기(3~60초) 적용
❖ 비영구적 사용 / 추가 비용 발생	❖ 영구적 사용 / 비용 절감
❖ 2차 인증(추가 인증)	❖ 2차 인증(추가 인증)
❖ 고가, 제한적 적용	❖ 저가, 다양하고 광범위한 적용
❖ 사용자 정보 동기화 필요	❖ 사용자 정보 동기화 필요하지 않음
❖ 인증 폭주 시 인증속도 저하	❖ 인증 폭주 시 부하분산으로 응답속도 보장
❖ 인증서버 장애 시 서비스 중단 발생	❖ 인증서버 장애 시 유연한 대처 (서비스 보장)

2.3 솔루션 개요

기업 및 개인의 정보 유출에 대한 해킹 피해 보도는 잊혀질 만 하면 계속 발생되고 있으며, 이에 대한 피해는 심각한 수준이다. 보다 근본적으로 해킹에 안전한 2차 인증키(일회용 인증키)를 사용하여 대응하여야 한다는 인식이 사회적으로 확산되고 있다.

BaroKEY 솔루션은 ERP, 전자결재, 그룹웨어 등 기업 내의 어플리케이션 로그인 시 비밀번호를 대체 또는 로그인-ID, 비밀번호 이외에 추가 인증(2차 인증)하는 소프트웨어 방식의 인증 솔루션으로, 일명 소프트 인증키, 2세대 인증키, 일회용 인증키로 불리며 H/W 방식의 인증키에 비해서 휴대성 및 편리성이 우수성이 뛰어난 솔루션이다.

1. 강력한 인증 솔루션

BaroKEY 솔루션 사용자 인증을 받기 위해서는 매번 새로운 **일회용 인증키**를 사용하여야 하며, 휘발성으로 1회에 한해서만 사용가능하며, ID/PW가 유출 시에도 안전하다.

한번 사용된 **일회용 인증키**는 재사용할 수 없으며, **일회용 인증키**의 유추가 어려워 다양한 해킹 공격에 강력한 보안성을 제공한다.

스마트 폰 기반인 BaroKEY는 단지 화면에 뜬 숫자를 입력하기만 하면 되는, 누구라도 손쉽게 사용가능한 단순한 구조이지만 기존에 사용하고 있던 고정된 ID/PW를 입력하는 것보다 더욱 안전하다.

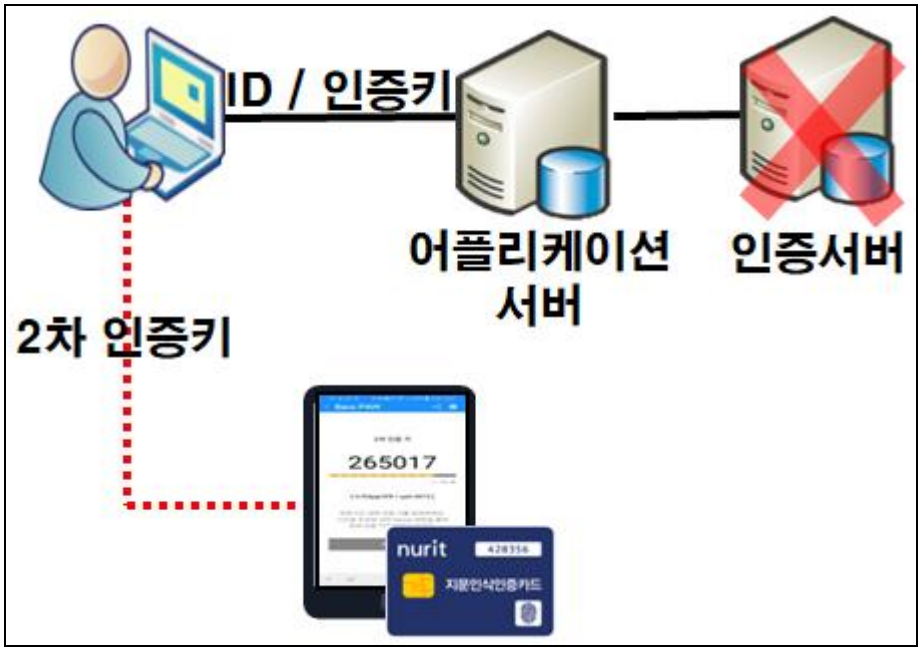
또한, PIN 번호 기능을 추가로 BaroPAM 앱이 설치된 스마트 폰 분실시에도 타인이 폰에 BaroPAM 앱에 등록된 정보를 알아내는 것은 불가능하므로 더욱 안전하게 사용할 수 있다.

2. 서비스의 용이성

2차 인증을 별도의 인증키 토큰 또는 카드 같은 별도의 인증키 장치를 휴대하지 않아도, 평소 휴대하고 다니는 사용자의 스마트 폰을 이용하여 인증을 처리하기 때문에 사용상의 편리성을 제공하며 휴대하기 간편하다.

ERP, 전자결재, 그룹웨어 등 기업 내의 어플리케이션 로그인시 사용자별 ID, 일회용 인증키 만으로 로그인 처리가 가능하므로 별도의 Password의 관리가 필요하지 않다.

3. 손쉬운 적용



BaroKEY 솔루션은 각 어플리케이션 서버에 일회용 인증키의 검증 모듈을 호출하는 구조(일회용 인증키의 검증 모듈은 jar, so, dll 현태로 제공)로 간단히 적용하며, 기존 네트워크 장비의 설정 변경이 불필요하다.

또한 별도의 인증서버가 필요 없는 구조(S/W 방식)로 관리, 운영 비용 및 H/W 인증키의 구매 비용을 절감할 수 있다.

2.4 솔루션 특/장점

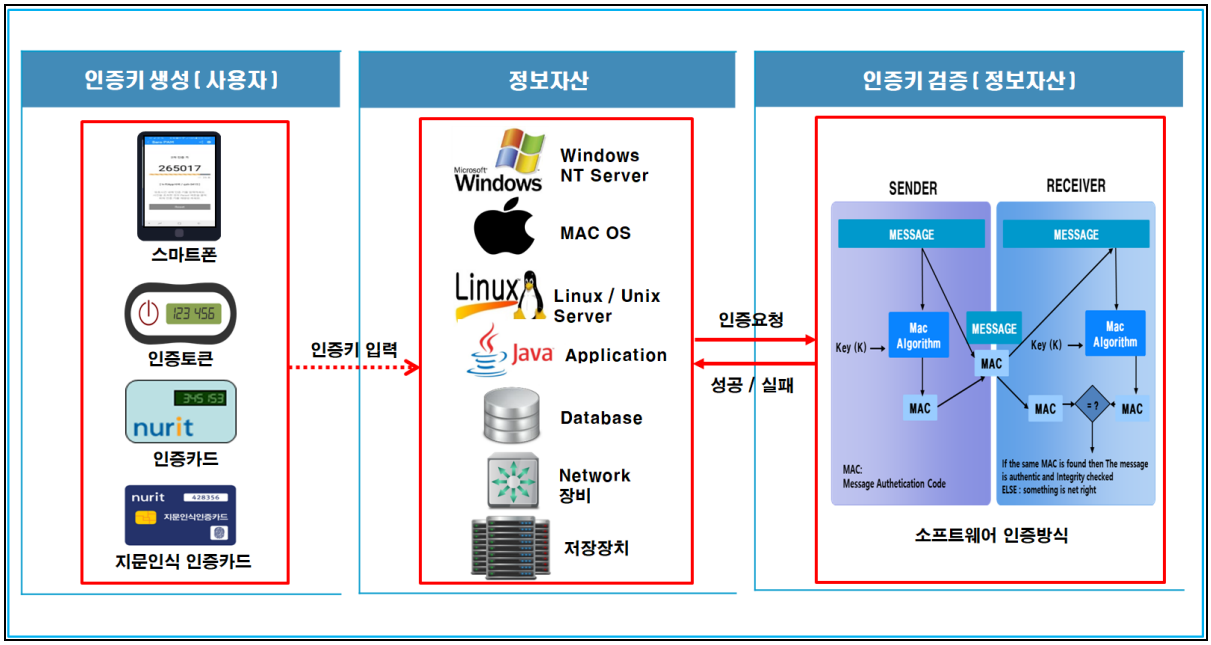
BaroKEY 솔루션은 2nd 인증키로써, 일회용 인증키의 생성기기 대신에 스마트폰(안드로이드, 아이폰)에 있는 일회용 인증키의 생성 모듈(소프트 인증키)로 일회용 인증키를 생성하여, 인증함으로써, ERP, 전자결재, 그룹웨어 등 기업 내의 어플리케이션 로그인 시 비밀번호 대체 및 추가 인증(2차 인증)을 위한 최적의 솔루션으로 특장점은 다음과 같다.

■ 전세계적으로 인정된 512bit 표준 해쉬 함수(HMAC-SHA512) 사용(인터넷 보안 표준인 IETF RFC 6283)
■ 금융감독원에서 권장하는 Time-Sync 방식
■ 동적 HMAC Key 방식
■ 어플리케이션, 도어락 등 사용자 인증이 필요한 모든 분야에서 사용 가능
■ Hard 인증키와는 달리 Soft 인증키(2nd 인증키)로 영구적 사용가능
■ 스마트 폰을 인증매체로 사용하므로 휴대성 및 편의성 극대화
■ 로그인-ID별로 인증Key 및 인증키 생성주기 개별 부여
■ 인증서버가 별도로 필요하지 않는 소프트웨어 방식(2세대 인증키, 소프트 인증키)
■ 2-Factor/2-Channel 인증의 비밀번호 대체 지원
■ 스마트 폰 분실 시 타인이 인증키 정보를 알아내는 것을 대비한 PIN 번호 기능 제공
■ 자유로운 Customizing 및 다양한 응용프로그램 및 연동 개발의 편의성 제공
(Java, C 언어로 된 API 연동)

※ HMAC (Hash-based Message Authentication Code) : 해쉬 기반 메시지 인증 코드
HMAC는 Key를 조합하여 Hash 함수를 구하는 방식으로, 송신자와 수신자만이 공유하고 있는 Key와 메시지를 혼합하여 Hash 값을 만드는 방식이다. 또한 채널을 통해 보낸 메시지가 훼손되었는지 여부를 확인하는데 사용할 수 있으며, MAC 특성상 역산이 불가능하므로, 수신된 메시지와 Hash 값을 다시 계산하여, 계산된 HMAC과 전송된 HMAC이 일치하는지를 확인하는 방식이다.

2.5 솔루션 구성

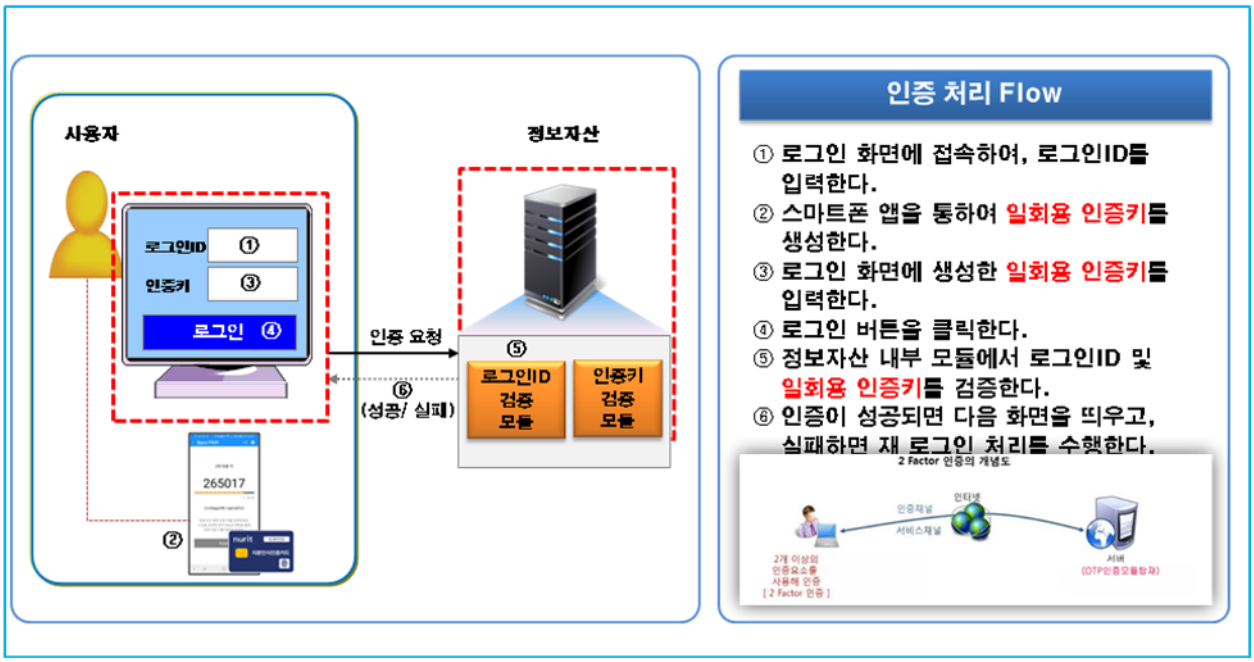
BaroKEY 솔루션은 사용자가 **일회용 인증키를 생성하는 장치**, **일회용 인증키를 적용하는 정보자산**, **일회용 인증키를 검증하는 부분**으로 구성되어 있습니다.



2.6 솔루션 처리 FLOW

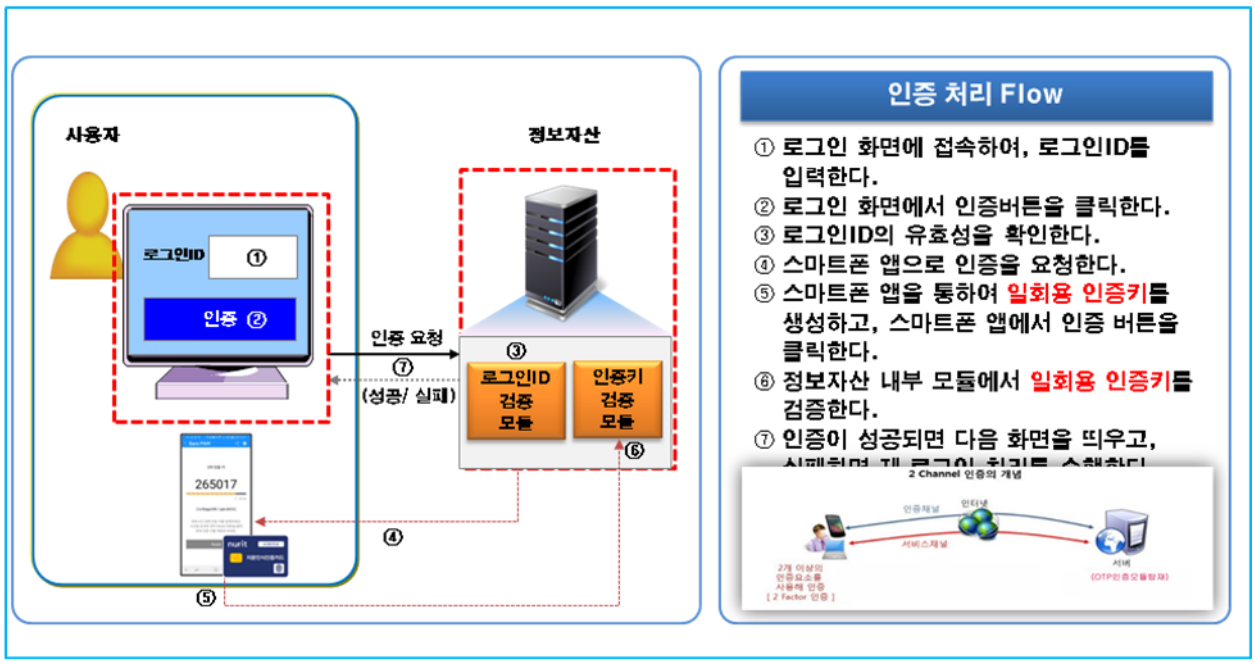
1. 2-Factor 인증 절차

BaroKEY 솔루션의 2-Factor 인증은 기존의 "**지식기반 인증**"인 ID/Password 인증에 **일회용 인증키**와 같은 2 개 이상의 인증 요소를 사용해 인증하는 **기법**으로 인증 처리 절차는 다음과 같다.



2. 2-Channel 인증 절차

BaroKEY 솔루션의 2-Channel 인증은 2-Factor 인증을 포함한다고 보는 것이 일반적이며, 인증과 서비스를 수행하는 통신선로를 **서비스 채널과 인증 채널로 물리적으로 분리하여 인증을 수행**한다.

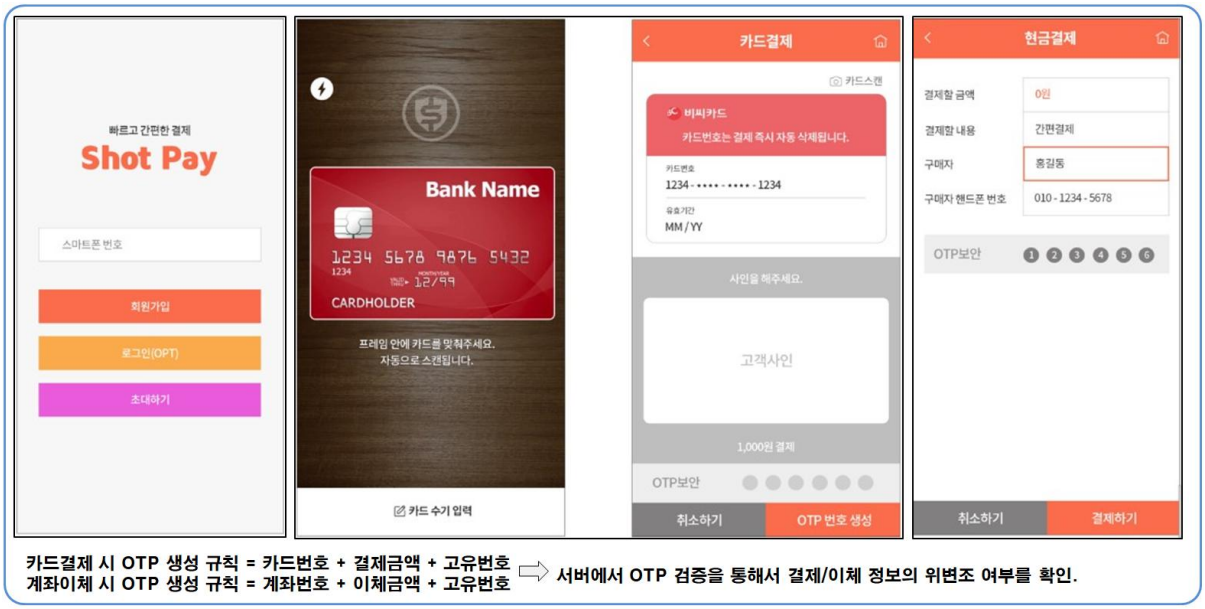


3. 어플리케이션 로그인

BaroKEY 솔루션의 정보자산 접근제어 2차 인증에 대한 시스템 Flow는 다음과 같이 처리된다.

4. 간편결제/계좌이체 인증(핀테크)

간편결제 및 계좌이체 시 앱에서 **OTP번호생성** 버튼을 클릭합니다. 생성한 OTP번호는 앱에 표시되며 결제버튼을 클릭하여 결제한다.



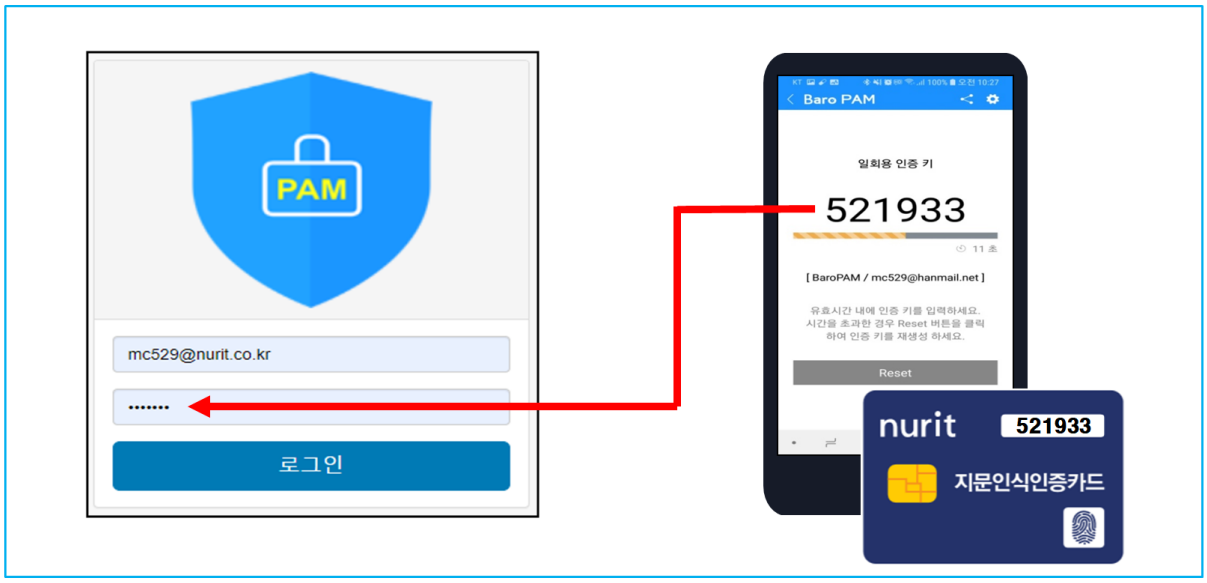
간편결제 및 계좌이체 시 OTP 생성 규칙을 적용하여 서버에서 OTP 검증을 통해서 카드결제/계좌이체 정보의 위변조 여부를 확인할 수 있다.

2.7 솔루션 적용분야

BaroKEY 솔루션은 정보자산, 도어락 등 사용자에 대한 2차 인증이 필요한 모든 분야에서 사용 가능하다.

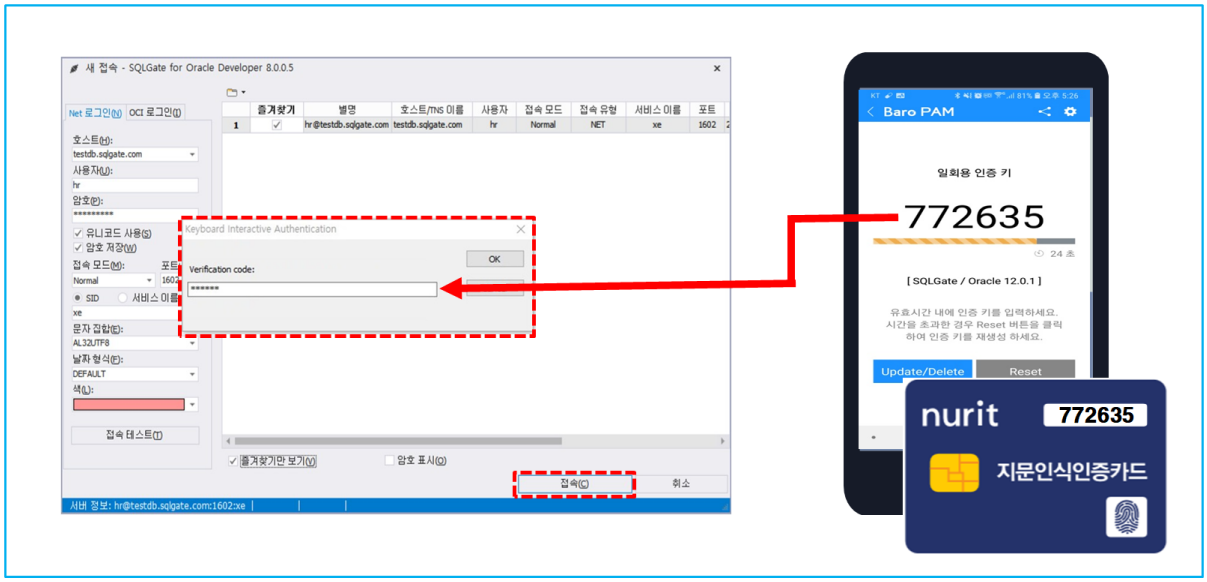
2.8 어플리케이션 로그인

어플리케이션 로그인 시 로그인-ID를 입력한 후 BaroPAM 앱에서 **일회용 인증키**를 생성한다. 생성한 **일회용 인증키**를 입력한 후 로그인 버튼을 클릭하여 어플리케이션에 로그인 한다.



2.9 DBMS 관리 툴 접속

Connection Wizard를 통한 DBMS 접속 시점에 **일회용 인증키**를 입력할 수 있는 "**Verification code**" 항목을 추가하여 보안을 강화한다.



2.10 SSO(Single Sign On) 솔루션과 융합

SSO(Single Sign On)는 1회 인증으로 여러 시스템을 이용할 수 있는 통합 인증 기능으로 단일화된 ID로 SSO로그인시 각 업무시스템을 별도의 인증절차 없이 권한에 따라 차등적, 선별적으로 각 시스템에 접근할 수 있는 환경을 제공하는 솔루션이다.

단일화된 ID로 SSO 로그인 시 로그인-ID를 입력한 후 스마트폰 앱에서 **일회용 인증키**를 생성한다. 생성한 **일회용 인증키** 입력하여 사용자의 한번 인증으로 연결된 모든 정보자산으로 접속 할 수 있는 통합 로그인과 융합하여 보완을 강화한다.

일회용 인증키 적용 전	일회용 인증키 적용 후
❖ 정보자산별 / 계정별 고정된 비밀번호 사용	❖ 정보자산별 / 계정별 일회용 인증키 사용
❖ 비밀번호 생성 규칙 적용	❖ 해시 알고리즘(SHA512)에서 발생한 값 적용
❖ 비밀번호 정보 DB에 보관	❖ 필요시 스마트폰 앱에서 직접 생성
❖ 암호화 기술 등을 이용한 보안 조치(단방향 암호화) 필요	❖ 암호화 기술 등을 이용한 보안 조치 불필요
❖ 유출 위험 및 피해 발생	❖ 유출 위험 및 피해 발생하지 않음
❖ 의무적으로 비밀번호 변경주기 적용	❖ 개별 인증키 생성주기(3~60초) 적용
❖ 비밀번호 증후군 / 비밀번호 리셋 증후군 호소	❖ 비밀번호 증후군 / 비밀번호 리셋 증후군 발생하지 않음

3. BaroKEY 연동 API

3.1 연동 API 구성

BaroKEY 관련 Shared object(libbarokey-6.1.1.so)는 필드 또는 데이터 암복호화 및 **일회용 인증키**를 검증하는데 사용된다.

API구분	설명	위치
barokey.h	BaroKEY Header 파일	/home/baropam/key
libbarokey-x.x.x.so	BaroKEY 모듈	/home/baropam/key

필드 또는 데이터 암복호화에 사용되는 대칭 키(64byte)를 프로그램 내부에 고정되어 있으며, Shared object(libbarokey-x.x.x.so)를 사용하기 위해서는 반드시 shared object file이 존재하는 디렉토리(/home/baropam/key)를 Library path에 설정 해야 한다.

```
Linux인 경우 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/baropam/key
HP-UX인 경우 export SHLIB_PATH=$SHLIB_PATH:/home/baropam/key
AIX인 경우   export LIBPATH=$LIBPATH:/home/baropam/key
```

BaroKEY에 대한 header 파일인 barokey.h은 다음과 같다.

```
#ifndef BAROKEY_INCLUDED
#define BAROKEY_INCLUDED

#include <stdlib.h>

#ifdef __cplusplus
extern "C" {
#endif

void * baro_encrypts(const void * data);
void * baro_decrypts(const void * data);

char *generateKEYL(const char *login_id, const char *phone_no, const char *cycle_time, const char *corr_time, const char *key_method);
char *generateKEYP(const char *secure_key, const char *cycle_time, const char *corr_time, const char *key_method);
int verifyKEYL(const char *login_id, const char *phone_no, const char *cycle_time, const char *corr_time, const char *key_method, char *tkey);
int verifyKEYP(const char *secure_key, const char *cycle_time, const char *corr_time, const char *key_method, char *tkey);

#ifdef __cplusplus
}
#endif

#endif
```

3.2 연동 API 함수

1) 암복호화 함수

① baro_encrypts 함수

- NAME
  baro_encrypts

- SYNOPSIS
  void * baro_encrypts(const void * data)

- DESCRIPTION
  데이터를 암호화 하는 함수
  data : 암호화할 데이터

- RETURN VALUES
  암호화한 데이터를 반환

② baro_decrypts 함수

- NAME
  baro_decrypts

- SYNOPSIS
  void * baro_decrypts(const void * data)

- DESCRIPTION
  데이터를 복호화 하는 함수
  data : 복호화할 데이터

- RETURN VALUES
  복호화한 데이터를 반환

③ 데이터 암복호화 사용 예

```
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "/home/baropam/key/barokey.h"

int main(int argc, char *argv[]) {
    const char *source_data = argv[1];
    const char *encrypt_data;
```

```
    const char *decrypt_data;

    encrypt_data = baro_encrypts(source_data );
    decrypt_data = baro_decrypts(encrypt_data);

    printf("Source   data = [%s]₩n", source_data );
    printf("encrypt data = [%s]₩n", encrypt_data);
    printf("decrypt data = [%s]₩n", decrypt_data);

    return 0;
}
```

2) 일회용 인증키 검증 함수

① verifyKEYL 함수(스마트폰 앱을 사용하는 경우)

- NAME
  verifyKEYL

- SYNOPSIS
  int verifyKEYL(const char *login_id, const char *phone_no, const char *cycle_time, const char *corr_time, const char *key_method, char *tkey)

- DESCRIPTION
  로그인 시 입력한 일회용 인증키가 맞는지 검증하는 함수

  login_id는 로그인 화면의 로그인-ID 항목에 입력한 ID를 설정해야 한다.
  phone_no는 사용자/고객/회원 정보에 있는 스마트 폰 번호를 숫자만 설정해야 한다.
  cycle_time은 사용자/고객/회원 정보에 개인별로 지정한 일회용 인증키의 생성 주기(3~60초)를 설정해야 한다.
  corr_time은 일회용 인증키의 보증오차시간(초)으로 인증카드인 경우만 설정해야 한다.(900초)
  key_method은 일회용 인증키의 생성 방식(app1, app256, app384, app512: 앱, card1, card256, card384, card512: 인증카드)을 설정해야 한다.
  tkey는 로그인 화면의 비밀번호에 입력한 일회용 인증키를 설정해야 한다.

  만약, 사용자/고객/회원 정보의 스마트 폰 번호 및 개인별로 지정한 일회용 인증키의 생성 주기가 일회용 인증키의 생성기와 다른 경우 일회용 인증키가 달라서 검증에 실패할 수 있다. 반드시 정보를 일치시켜야 한다.

- RETURN VALUES
  성공 시에는 1을 반환하며, 실패 시는 0을 반환한다.

② verifyKEYP 함수(인증카드를 사용하는 경우)

- NAME
  verifyKEYL

- SYNOPSIS
  int verifyKEYL(const char *login_id, const char *phone_no, const char *cycle_time, const char *corr_time, const char *key_method, char *tkey)

- DESCRIPTION
  로그인 시 입력한 **일회용 인증키**가 맞는지 검증하는 함수

  secure_key는 사용자/고객/회원 정보에 있는 Secure key(개별 부여)를 설정해야 한다.
  cycle_time은 사용자/고객/회원 정보에 개인별로 지정한 **일회용 인증키**의 생성 주기(3~60초)를 설정해야 한다.
  corr_time은 **일회용 인증키**의 보증오차시간(초)으로 인증카드인 경우만 설정해야 한다.(900초)
  key_method은 **일회용 인증키**의 생성 방식(app1, app256, app384, app512: 앱, card1, card256, card384, card512: 인증카드)을 설정해야 한다.
  tkey는 로그인 화면의 비밀번호에 입력한 **일회용 인증키**를 설정해야 한다.

  만약, 사용자/고객/회원 정보의 스마트 폰 번호 및 개인별로 지정한 **일회용 인증키**의 생성 주기가 **일회용의 인증키**의 생성기와 다른 경우 **일회용 인증키**가 달라서 검증에 실패할 수 있다. 반드시 정보를 일치시켜야 한다.

- RETURN VALUES
  성공 시에는 1을 반환하며, 실패 시는 0을 반환한다.

3.3 인증키 검증 부분

1) 일회용 인증키의 검증 모듈 사용 예(Linux 환경)

Sample program) verifyKEYL 함수(스마트폰 앱을 사용하는 경우)

```
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "/home/baropam/key/barokey.h"

int main(int argc, char *argv[]) {
    const char *login_id   = argv[1];    // 로그인-ID
    const char *phone_no   = argv[2];    // 폰번호
    const char *cycle_time = argv[3];    // 생성주기
    const char *corr_time  = argv[4];    // 보정시간
    const char *key_method = argv[5];    // 인증키 생성방식
          char *tkey       = argv[6];    // 일회용 인증키

    // 일회용 인증키 검증
    int  bkey = verifyKEYL(login_id, phone_no, cycle_time, corr_time, key_method, tkey);

    // 일회용 인증키 검증(성공)
    if (bkey == 1) {
        printf("Auth key success.Wn");
```

```
    // 일회용 인증키 검증(실패)
    } else {
        printf("Auth key faild.₩n");
    }
}
```

Sample program) verifyKEYP 함수(인증카드를 사용하는 경우)

```
#include <errno.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "/home/baropam/key/barokey.h"

int main(int argc, char *argv[]) {
    const char *secure_key = argv[1];    // Secure key
    const char *cycle_time = argv[2];    // 생성주기
    const char *corr_time  = argv[3];    // 보정시간
    const char *key_method = argv[4];    // 인증키 생성방식
          char *tkey       = argv[5];    // 일회용 인증키

    // 일회용 인증키 검증
    int  bkey = verifyKEYP(secure_key, cycle_time, corr_time, key_method, tkey);

    // 일회용 인증키 검증(성공)
    if (bkey == 1) {
        printf("Auth key success.₩n");
    // 일회용 인증키 검증(실패)
    } else {
        printf("Auth key faild.₩n");
    }
}
```

컴파일)

```
$ gcc -o barokeys barokeys.c -L/home/baropam/key -lbarokey-x.x.x
```

실행)

```
$ ./barokeys mc529@hanmail.net 01027714076 20 0 app512 490661
Auth key success.
```

참고) 실행시 다음과 같은 오류가 발생하는 경우

```
./barokeys: error while loading shared libraries: libbarokey.so: cannot open shared object file:
No such file or directory
```

이런 경우 shared object file이 존재하지 않거나 shared object file가 존재하는 디렉토리가 Library path에 설정되어 있지 않아서 발생하므로 shared object file이 존재여부를 확인하고 shared object file가 존재하는 디렉토리(/home/baropam/key)를 Library path에 설정해야 한다.

```
Linux인 경우 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/baropam/key
HP-UX인 경우 export SHLIB_PATH=$SHLIB_PATH:/home/baropam/key
AIX인 경우   export LIBPATH=$LIBPATH:/home/baropam/key
```

2) 일회용 인증키의 검증 모듈 사용 예(Tuxedo 환경)

예) verifyKEYL 함수(스마트폰 앱을 사용하는 경우)

```
#include <stdio.h>
#include <ctype.h>
#include <atmi.h>       /* TUXEDO Header File */
#include <userlog.h>    /* TUXEDO Header File */

#include "/home/baropam/key/barokey.h"

TOUPPER(TPSVCINFO *rqst) {
    const char *login_id   = "mc529@hanmail.net";
    const char *phone_no   = "01027714076";
    const char *cycle_time = "20";
    const char *corr_time  = "0";
    const char *key_method = "app512";
          char *tkey       = rqst->data;

    strcpy(tkey, rqst->data);
    int  bkey = verifyKEYL(login_id, phone_no, cycle_time, corr_time, key_method, tkey);

    sprintf(rqst->data, "%d", bkey);
    /* Return the transformed buffer to the requestor. */
    tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
}
```

예) verifyKEYP 함수(인증카드를 사용하는 경우)

```
#include <stdio.h>
#include <ctype.h>
#include <atmi.h>       /* TUXEDO Header File */
#include <userlog.h>    /* TUXEDO Header File */

#include "/home/baropam/key/barokey.h"

TOUPPER(TPSVCINFO *rqst) {
    const char *secure_key = "Ri5+xgVdtEBJGlrSD2hvituZxAq0vttx";
    const char *cycle_time = "20";
    const char *corr_time  = "900";
    const char *key_method = "card512";
          char *tkey       = rqst->data;

    strcpy(tkey, rqst->data);
    int  bkey = verifyKEYP(secure_key, cycle_time, corr_time, key_method, tkey);

    sprintf(rqst->data, "%d", bkey);
    /* Return the transformed buffer to the requestor. */
```

```
    tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
}
```

컴파일)

```
$ CFLAGS="-L/home/baropam/key -lbarokey-x.x.x" buildserver -v -o simpserv -f simpserv.c -s TOUPPER
```

참고) Tuxedo 서비스에서 **일회용 인증키**를 검증하는 경우, 컴파일 시 CFLAGS 옵션에 일회용 검증키 모듈을 "**-L/home/baropam/key -lbarokey-x.x.x**" 설정해야 한다.

Tuxedo 서버 프로세스 기동시 다음과 같은 오류가 발생하는 경우

```
error while loading shared libraries: libbarokey-x.x.x.so: cannot open shared object file: No such file or directory
```

이런 경우 shared object file이 존재하지 않거나 shared object file가 존재하는 디렉토리가 Library path에 설정되어 있지 않아서 발생하므로 shared object file이 존재여부를 확인하고 shared object file가 존재하는 디렉토리(/home/baropam/key)를 Library path에 설정해야 한다.

```
Linux인 경우 export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/baropam/key
HP-UX인 경우 export SHLIB_PATH=$SHLIB_PATH:/home/baropam/key
AIX인 경우   export LIBPATH=$LIBPATH:/home/baropam/key
```

4. NTP(Network Time Protocol) 설정

최근에는 서버/네트워크 장비에 대한 시간 동기화(타임서버 시간 동기화)하는 방법으로 NTP(Network Time Protocol)을 이용하여 관리자 계정에서 시스템의 시각을 현재 시각으로 설정할 수 있다.

4.1 Windows 환경

1. NTP란?

Network Time Protocol(네트워크 시간 프로토콜)의 약자로 네트워크 환경으로 구성된 장비(서버, PC, 통신장비, 방화벽 장비 등)의 시스템 시간을 동기화 하기 위한 규약이다.

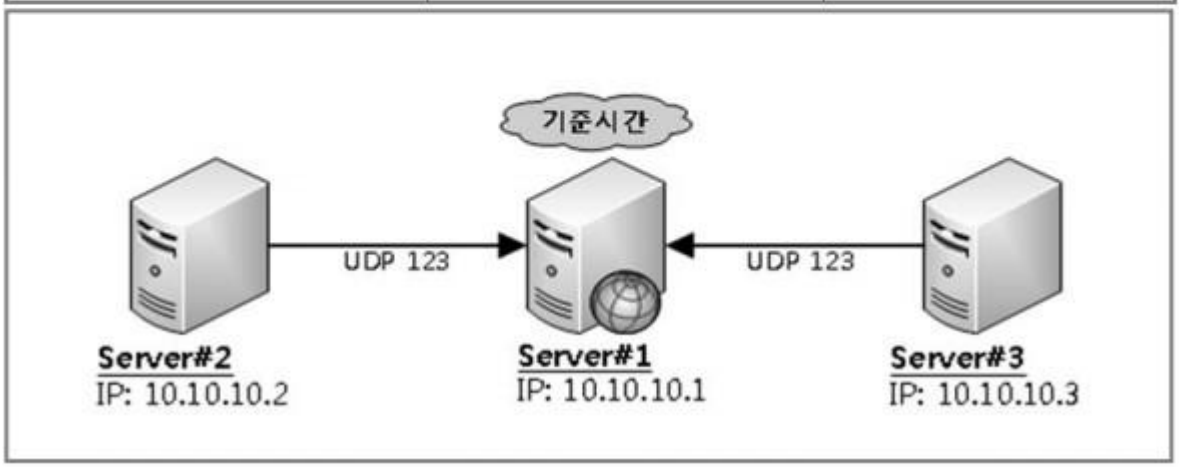
윈도우에는
- 모든 윈도우서버는 NTP 서버가 될 수 있다.
- 모든 서버의 방화벽에서 UDP 123 번이 열려있어야 한다.
- NTP 서버와의 시간차이가 많이 날 경우(기본값: 15 시간) 동기화 되지 않는다.
  실제 시간과 근접한 시간으로 변경 후 동기화 할 수 있도록 한다.
- 데이터 일치를 위하여 동기화(Sync)요청을 하더라도 즉시 반영되는 것이 아니라, 조금씩 맞춰 간다.
  (예: Server#2 서버가 Server#1 서버 보다 시간이 빠를 경우, Server#1 번의 시간으로 Server#2 의 시스템시간을 동기화 할 경우, 동일한 시간대의 데이터가 생성되므로, 데이터 정합성에 오류가 발생한다. 이에 시간 동기화는 즉시 반영되는 것이 아니라, 조금씩 그 차이를 줄여나가는 방식이다. - 표 12 참조)

2. 시스템 환경

NTP 시간을 제공해주는 서버를 “NTP 서버”, NTP 서버로 시간 동기화를 요청하는 서버를 “Slave 서버”로 명명한다.

서버명 및 IP 주소

서버명	IP 주소	역할
Server#1	10.10.10.1	NTP 서버
Server#2	10.10.10.2	Slave 서버
Server#3	10.10.10.3	Slave 서버



3. 내부 NTP 서버

1) 외부 NTP 서버와 시간 동기화

내부 NTP 서버 시스템 시간을 인터넷 표준시(예:time.bora.net)으로 설정한다.

① 실행서버
Server#1(IP:10.10.10.1)

② 시나리오
내부 NTP 서버 역할로 운용할 서버의 시간 동기화 대상을 외부 NTP(예:time.bora.net, time.kornet.net)으로 설정하고, 설정(레지스트리)값을 확인 후 동기화 작업을 수행한다.

③ 작업
첫번째, 외부 NTP 서버를 기준으로 시간동기화 설정을 한다.
* 시작 -> cmd -> w32tm /config /syncfromflags:manual /manualpeerlist:time.microsoft.com /update

표 1: 예상결과

```
C:₩>w32tm /config /syncfromflags:manual /manualpeerlist:time.microsoft.com /update
명령이 성공적으로 완료되었습니다
```

두번째, Windows Time 서비스의 설정값(레지스트리) 확인
* 시작 -> cmd -> w32tm /dumpreg /subkey:Parameters

표 2: 예상 결과

```
C:₩>w32tm /dumpreg /subkey:Parameters
값 이름         값 종류           값 데이터
------------------------------------------------------------------------------

ServiceMain     REG_SZ            SvchostEntry_W32Time
ServiceDll      REG_EXPAND_SZ     C:₩WINDOWS₩system32₩w32time.dll
NtpServer       REG_SZ            time.bora.net,0x9 <- 외부 NTP 서버주소
Type            REG_SZ            NTP                  <- 외부 NTP 서버 사용
```

세번째, Windows Time 서비스 재시작
* 시작 -> cmd -> net stop w32tm
-> net start w32tm

표 3: 예상결과

```
C:₩>net stop w32time
Windows Time 서비스를 멈춥니다..
Windows Time 서비스를 잘 멈추었습니다.

C:₩>net start w32time
Windows Time 서비스를 시작합니다..
Windows Time 서비스가 잘 시작되었습니다.
```

네번째, 시간 동기화
* 시작 -> cmd -> w32tm /resync

표 4: 예상결과

```
[정상]
C:₩>w32tm /resync
로컬 컴퓨터에 다시 동기화 명령을 보내는 중
```

```
명령이 성공적으로 완료되었습니다.

[오류] : 방화벽에서 외부 UDP 123 포트가 막혀 있을 경우 발생할 수 있다.
C:W>w32tm /resync
동기화 명령 전송 - local computer...
사용 가능한 시간 데이터가 없어 컴퓨터가 동기화하지 못했습니다.
```

2) NTP 서비스 제공을 위한 구성 확인

내부 NTP 서버가 외부 NTP 서버를 참조하는지 확인한다.

① 실행서버
Server#1(IP:10.10.10.1)

② 시나리오
내부 NTP 서버 역할로 운용할 서버의 서비스를 확인하여 Slave 서버에서 연결 할 수 있도록 구성 값을 확인한다.

③ 작업
첫번째, Windows Time 서비스 구동 확인
* 시작 -> cmd -> sc query w32time

표 5: 예상 결과

```
C:W>sc query w32time
        SERVICE_NAME      : w32time
        TYPE              : 20 WIN32_SHARE_PROCESS
        STATE             : 4 RUNNING
                            (STOPPABLE,NOT_PAUSABLE,ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE   : 0 (0x0)
        SERVICE_EXIT_CODE : 0 (0x0)
        CHECKPOINT        : 0x0
        WAIT_HINT         : 0x0
```

두번째, NTP 서비스 구동 확인
* 시작 -> cmd -> netstat -ano | findstr 123

표 6: 예상 결과

```
C:W>netstat -ano | findstr 123
  UDP    0.0.0.0:123      *:*     1128
  UDP    0.0.0.0:62123    *:*     1428
  UDP    [::]:123         *:*     1128
```

4. Slave 서버

1) Slave 서버 환경 구성

내부 NTP 서버(IP:10.10.10.1)를 기준으로 Slave 서버의 시스템 시간을 설정한다. 시간 차이가 너무 많을 경우 동기화가 되지 않으니, 수동으로 근접한 시간을 맞추고 동기화 할 수 있도록 한다.

① 실행서버
Server#2(IP:10.10.10.2)
Server#3(IP:10.10.10.3)

(각 서버별로 수행)

② 시나리오
시간 동기화 대상을 내부 NTP 서버(IP:10.10.10.1)로 설정하고, 동기화 작업을 수행한다.

③ 작업
첫번째, 내부 NTP 서버(IP:10.10.10.1)를 기준으로 시간동기화 설정을 한다.
* 시작 -> cmd -> w32tm /config /syncfromflags:manual /manualpeerlist:10.10.10.1 /update

표 7: 예상결과

```
C:₩>w32tm /config /syncfromflags:manual /manualpeerlist:10.10.10.1 /update
명령이 성공적으로 완료되었습니다.
```

두번째, Windows Time 서비스의 설정값(레지스트리) 확인
* 시작 -> cmd -> w32tm /dumpreg /subkey:Parameters

표 8: 예상 결과

```
C:₩>w32tm /dumpreg /subkey:Parameters
값 이름         값 종류           값 데이터
----------------------------------------------------------------------

ServiceMain     REG_SZ          SvchostEntry_W32Time
ServiceDll      REG_EXPAND_SZ    C:₩WINDOWS₩system32₩w32time.dll
NtpServer       REG_SZ          10.10.10.1 <- 외부 NTP 서버주소
Type            REG_SZ          NTP        <- 외부 NTP 서버 사용
```

세번째, Windows Time 서비스 재시작
* 시작 -> cmd -> net stop w32tm
-> net start w32tm

표 9: 예상결과

```
C:₩>net stop w32time
Windows Time 서비스를 멈춥니다..
Windows Time 서비스를 잘 멈추었습니다.

C:₩>net start w32time
Windows Time 서비스를 시작합니다..
Windows Time 서비스가 잘 시작되었습니다.
```

세번째, 시간 동기화
* 시작 -> cmd -> w32tm /resync

표 10: 예상결과

```
C:₩>w32tm /resync
로컬 컴퓨터에 다시 동기화 명령을 보내는 중
명령이 성공적으로 완료되었습니다.
```

2) NTP 서버와 시간차 확인

특정 서버(ex.NTP 서버)와 실행한 서버와의 시간차이를 확인한다.

① 실행서버
Server#2(IP:10.10.10.2)

Server#3(IP:10.10.10.3)
(각 서버별로 수행하여 확인 할 수 있다.)

② 시나리오
특정서버(Master Server:10.10.10.1)와 Slave Server(10.10.10.2)시간 동기화 대상을 Master Server 설정하고, 동기화 작업을 수행한다.

③ 작업
첫번째, Master Server 를 기준으로 시간동기화 설정을 한다.
* 시작 -> cmd -> w32tm /stripchart /dataonly /computer:10.10.10.1
  비교대상 서버와 시간이 일치한다면, 하기와 유사한 결과값을 볼 수 있다.

표 11: 예상결과

```
C:₩> w32tm /stripchart /dataonly /computer:10.10.10.1
Tracking 10.10.10.1 [10.10.10.1].
The current time is 2012-04-18 오후 19:17:13 (local time).
19:17:13, +00.2676328s     -> 비교 대상 서버와 +00.2676328 초만큼 차이가 난다.
19:17:15, +00.2593851s
19:17:17, +00.2589499s
19:17:19, +00.2428931s
^C                         -> 중지하기 위해서는 "Ctrl + c"를 누르면 된다.
      +00.xxxxxxx(또는 -00.xxxxxxx)로 대상 NTP 서버와 차이나는 시간만큼 표기된다.
      예) +120.2428931s -> +120 초 차이가 남
```

5. NTP 설정 후 동기화 요청 및 동기화 확인

표 12: Slave 서버 동기화 설정

NTP 서버(IP:10.10.10.1) 서버를 시스템 시간으로 설정한다.

```
C:₩>w32tm /config /syncfromflags:manual /manualpeerlist:10.10.10.1/update
명령이 성공적으로 완료되었습니다.
```

Windows Time 서비스를 재기동한다.

```
C:₩>net stop w32time
Windows Time 서비스를 멈춥니다..
Windows Time 서비스를 잘 멈추었습니다.

C:₩>net start w32time
Windows Time 서비스를 시작합니다..
Windows Time 서비스가 잘 시작되었습니다.
```

NTP 서버(IP:10.10.10.1)로 지정한 서버와 동기화를 요청한다.

```
C:₩>w32tm /resync
로컬 컴퓨터에 다시 동기화 명령을 보내는 중
명령이 성공적으로 완료되었습니다.
```

NTP 서버(IP:10.10.10.1)와 실행 중인 서버와의 시간차이를 확인한다.

```
C:₩>w32tm /stripchart /dataonly /computer:10.10.10.1
10.10.10.1[10.10.10.1:123] 추적 중
현재 시간은 2012-04-19 오후 22:47:25 입니다.
22:47:25, +23.2364090s
22:47:27, +22.7004942s     -> 조금씩이지만 차이가 줄어드는 것을 볼 수 있다.
22:47:30, +22.1639462s
```

```
22:47:32, +21.6430236s
^C                          -> 중지하기 위해서는 "Ctrl + c"를 누르면 된다.
```

6. 주기적인 시간 동기화

주기적으로 시간을 자동으로 동기화 하게 만드는데, 방법은 다음과 같다.

1) 시작 - 실행 - regedit (혹은 Winkey+R - regedit)

2) HKEY_LOCAL_MACHINE₩
   SYSTEM₩
   CurrentControlSet₩
   services₩
   W32Time₩
   TimeProviders₩
   NtpClient

3) SpecialPollInterval 이름을 더블 클릭

4) 10진수로 바꿔서 본인이 원하는 숫자를 입력.
   60 = 60초, 600 = 10분, 3600 = 1시간, 86400 = 1일
   초로 계산한다.

5) 시작 - 실행 - cmd (혹은 Winkey+R - cmd)

6) Windows Time 서비스 재기동
   net stop w32time 엔터
   net start w32time 엔터

4.2 Linux 환경

최근에는 Windows/서버/데이터베이스/네트워크 장비/저장장치에 대한 시간 동기화(타임서버 시간 동기화) 하는 방법으로 NTP(Network Time Protocol)을 이용하여 루트 계정에서 시스템의 시각을 현재 시각으로 설정할 수 있다.

NTP를 사용하기 위해서는 기본적으로 NTP 패키지가 반드시 설치되어 있어야 한다. 설치 확인은 다음의 명령어를 실행하여 확인한다. 만약, 설치되어 있지 않으면 Redhat, CentOS 8 이하 버젼은 "**yum install ntp**" 그외는 "**sudo apt-get install ntp**" 명령어로 설치하면 된다.

```
[root]# rpm -qa | grep ntp
ntp-4.2.2p1-18.el5.centos
chkfontpath-1.10.1-1.1
```

ntpd 서비스를 서버 부팅 시 시작프로그램에 등록 및 ntp 활성화 여부 확인은 다음과 같은 명령어로 확인할 수 있다.

```
[root]# chkconfig ntpd on
[root]# chkconfig —list | grep ntp
ntpd            0:해제  1:해제  2:활성  3:활성  4:활성  5:활성  6:해제
```

chkconfig 이용하여 서버 부팅시 ntpd 데몬 활성화 여부 확인 3, 5 level 에 off(해제) 가 되어 있으면 자동 활성화 되지 않는다. 자동 활성화 하기 위해서는 3, 5 에 on(활성)으로 다음과 같은 명령어로 변경해야 한다.

```
[root]# chkconfig —level 3 ntpd on
[root]# chkconfig —level 5 ntpd on
```

우리나라에서 운영되고 있는 NTP 서버는 다음과 같다.

```
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net
```

우리나라에서 운영되고 있는 NTP 서버를 ntpd 데몬 설정을 위한 설정 파일인 "/etc/ntp.conf"에 다음과 같이 설정한다.

```
[root]# vi /etc/ntp.conf
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#server 0.centos.pool.ntp.org
#server 1.centos.pool.ntp.org
#server 2.centos.pool.ntp.org
#server 3.centos.pool.ntp.org
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net
```

ntpd 데몬 설정을 위한 설정이 끝나면 반드시 NTP 설정이 제대로 추가되었는지 확인한 후 NTP 데몬의 Restart 작업이 반드시 필요하다.

```
[root]# /etc/init.d/ntpd restart
ntpd를 종료 중: [  OK  ]
ntpd (을)를 시작 중: [  OK  ]
```

ntpd 시간 확인은 다음과 같은 명령어로 확인할 수 있다.

```
[root]# ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 static.betaidc. 106.247.248.106  3 u    7   64    1    2.884  287.718   0.001
 time.bora.net   .INIT.          16 u    -   64    0    0.000    0.000   0.000
 183.110.225.61  .INIT.          16 u    -   64    0    0.000    0.000   0.000
 LOCAL(0)        .LOCL.          10 l    4   64    1    0.000    0.000   0.001
```

* 표시된 ip 가 현재 시간을 가져오고 있는 ntp 서버임

NTP를 사용하기 위해서는 기본적으로 NTP 패키지가 반드시 설치되어 있어야 한다. 설치 확인은 다음의 명령어를 실행하여 확인한다. 만약, 설치되어 있지 않으면 Redhat, CentOS 8 이상 버젼은 "yum install chrony" 명령어로 설치하면 된다.

```
[root@baropam ~]# rpm -qa | grep chrony
chrony-3.5-1.el8.x86_64
```

우리나라에서 운영되고 있는 NTP 서버는 다음과 같다.

```
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net
```

우리나라에서 운영되고 있는 NTP 서버를 ntpd 데몬 설정을 위한 설정 파일인 “/etc/chrony.conf”에 다음과 같이 설정한다.

```
[root@baropam ~]# vi /etc/chrony.conf

# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#pool 2.centos.pool.ntp.org iburst
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net

# Record the rate at which the system clock gains/losses time.
driftfile /var/lib/chrony/drift

# Allow the system clock to be stepped in the first three updates
# if its offset is larger than 1 second.
makestep 1.0 3

# Enable kernel synchronization of the real-time clock (RTC).
rtcsync

# Enable hardware timestamping on all interfaces that support it.
#hwtimestamp *

# Increase the minimum number of selectable sources required to adjust
# the system clock.
#minsources 2

# Allow NTP client access from local network.
allow 192.168.0.0/16

# Serve time even if not synchronized to a time source.
#local stratum 10

# Specify file containing keys for NTP authentication.
keyfile /etc/chrony.keys

# Get TAI-UTC offset and leap seconds from the system tz database.
leapsectz right/UTC

# Specify directory for log files.
logdir /var/log/chrony

# Select which information is logged.
#log measurements statistics tracking
```

ntpd 데몬 설정을 위한 설정이 끝나면 반드시 NTP 설정이 제대로 추가되었는지 확인한 후 NTP 데몬의

Restart 작업이 반드시 필요하다.(chrony 서비스 시작 및 부팅시 구동 등록)

```
[root@baropam ~]# systemctl start chronyd
[root@baropam ~]# systemctl enable chronyd
```

ntpd 시간 확인은 다음과 같은 명령어로 확인할 수 있다.

시간을 받아오는 서버 리스트 / chrony.conf 파일에 등록한 server 리스트)

```
[root@baropam ~]# chronyc sources
210 Number of sources = 2
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* ec2-54-180-134-81.ap-nor>     2   6   377    43   -349us[-1059us] +/-   24ms
^- time.bora.net                 2   6   377    42  +1398us[+1398us] +/-   90ms
```

시간을 받아 오는 서버 정보)

```
[root@baropam ~]# chronyc tracking
Reference ID    : 36B48651 (ec2-54-180-134-81.ap-northeast-2.compute.amazonaw)
Stratum         : 3
Ref time (UTC)  : Sun Mar 22 07:07:43 2020
System time     : 0.000130027 seconds slow of NTP time
Last offset     : -0.000710122 seconds
RMS offset      : 0.000583203 seconds
Frequency       : 19.980 ppm fast
Residual freq   : +0.142 ppm
Skew            : 3.235 ppm
Root delay      : 0.013462566 seconds
Root dispersion : 0.017946836 seconds
Update interval : 65.0 seconds
Leap status     : Normal
```

시간 상태 및 동기화 등 정보 확인)

```
[root@baropam ~]# timedatectl status
               Local time: Sun 2020-03-22 16:08:45 KST
           Universal time: Sun 2020-03-22 07:08:45 UTC
                 RTC time: Sun 2020-03-22 07:08:44
                Time zone: Asia/Seoul (KST, +0900)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

4.3 Solaris 환경

NTP(Network Time Protocol)는 컴퓨터 클라이언트나 서버의 시간을 다른 서버나 라디오 또는 위성 수신기와 같은 참조할 수 있는 타임 소스 또는 모뎀에 동기화하는데 사용된다.

1) 세 가지 유형의 시간 서버(Time Server)

① peer host_address [key #] [version #] [prefer]
- 로컬 서버가 호스트 주소로서 지정된 원격 서버와 함께 symmetric active 모드로 운영.
- 로컬 서버는 원격 서버에 맞추어 동기화 할수 있음.

② server host_address [kery #] [version #] [prefer] [mode #] server
- 로컬 서버가 command에서 이름이 지정된 원격 서버와 함께 client 모드로 운영.
- 이 모드에서 로컬 서버는 원격 서버에 맞추어 동기화될 수 있지만 원격 서버는 로컬 서버에 동기화 할 수 없음.

③ broadcast host_address [key#] [version #] [ttl #]
로컬 서버가 broadcast 모드로 운영된다는 것을 지정한다. 이 모드에서 로컬 서버는 명령어에서 지정된 broadcast/multicast 주소의 클라이언트 무리에게 정기 적인 broadcast 메시지를 전송한다.
- key 주소에 전송된 모든 패킷이 지정된 키 번호를 사용하여 암호화된 인증 필드를 포함
- version outgoing NTP 패킷에 사용되는 버전 번호를 지정 Version ①, ②, ③ 선택 기본 버전 ③번.
- prefer 호스트를 선택된 호스트로 표시,동기화를 위해 다른 비교 가능한 호스트보다 이 호스트가 선택.

2) NTP SERVER 설정. (server 모드) 방법

```
(sun>root)/etc/inet# cp ntp.server ntp.conf
(sun>root)/etc/inet# vi ntp.conf
# Either a peer or server.  Replace "XType" with a value from the
# table above.
#server 127.127.XType.0 prefer
#fudge 127.127.XType.0 stratum 0
server 127.127.1.0 ---> 언제나 로컬로 돌아갈수 있음.
server time.kriss.re.kr prefer
server 127.127.1.0
server gps.bora.net
server ntp.ewha.net
server time.bora.net
server time.nuri.net
server ntp2.gngidc.net
server time.kriss.re.kr

#broadcast 224.0.1.1 ttl 4 -->기본 설정 ( 네트웍내 여러개의 ntp 서버가 존재할 경우 변경)
broadcast 192.168.0.222 ttl 4
wq!
```

3) NTP SERVER 설정. (피어(peer) 모드) 방법

peer gps.bora.net key 0 version 3 prefer --> server 모드와 동일 server 설정대신 peer 설정.

```
(sun>root)/etc/inet# /etc/init.d/xntpd start
(sun>root)/etc/inet# ps -ef | grep ntp
    root  479  400  0 08:40:55 pts/2    0:00 grep ntp
    root  456    1  0 08:23:07 ?       0:01 /usr/lib/inet/xntpd
(sun>root)/etc/inet# ntpq -p (NTP 서버에게 피어 리스트에 관해 질의)
     remote           refid      st t when poll reach   delay   offset    disp
==============================================================================
 sun            0.0.0.0         16 -    -  64    0    0.00    0.000 16000.0
```

```
 gps.bora.net    0.0.0.0         16 u  48  64    0    0.00    0.000 16000.0
(e220>root)/# snoop -d hme0 port 123
Using device /dev/hme (promiscuous mode)
192.168.0.222 -> gps.bora.net NTP  symmetric active (Fri Feb 24 08:35:26 2006)
gps.bora.net -> 192.168.0.222 NTP  server (Sat Jan 19 03:58:31 2002)
       e220 -> 192.168.0.222 NTP  client (Fri Feb 24 08:35:46 2006)
192.168.0.222 -> e220        NTP  server (Fri Feb 24 08:35:47 2006)
```

4) NTP Client 설정

```
(sun>root)/etc/inet# cp ntp.client ntp.conf
```

기본 ntp.client 파일은 multicast를 사용하여 ntp 업데이트를 수신한다. NTP 클라이언트가 이러한 업데이트를 수신할 수 있는 장소를 제한하려는 경우 이것을 broadcast로 변경한다.(broadcast 패킷은 다른 서브넥에 전달되지 않는 반면 multicast 패킷은 전달 된다.)

```
(sun>root)/etc/inet# /etc/init.d/xntpd start
#multicastclient 224.0.1.1 —> 기본설정
server 192.168.0.222
wq!

(sun>root)/etc/inet# ps -ef | grep ntp
    root  479  400  0 08:40:55 pts/2    0:00 grep ntp
    root  456    1  0 08:23:07 ?       0:01 /usr/lib/inet/xntpd
(sun>root)/etc/inet#
(sun>root)/etc/inet# ntpq -p
    remote         refid      st t when poll reach  delay  offset    disp
==============================================================================
 sun           0.0.0.0         16 -    -  64    0    0.00    0.000 16000.0
 gps.bora.net    0.0.0.0         16 u  48  64    0    0.00    0.000 16000.0
```

remote-원격 피어, refid-피어가 동기화되는 호스트, st-stratum 번호, t-유형 즉 unicast, multcst, local(- = 알수 없음), poll-초 단위 폴링 간격, reach-도달 가능성 레지스터
* 원격에서 현재 선택된 피어를 나타낸다.
+ 호스트가 동기화에 대한 수락 가능한 피어이지만 수락되지 않았음을 나타냄.
_ 수락 불가능

4.4 HP-UX 환경

최근에는 서버/네트워크 장비에 대한 시간 동기화(타임서버 시간 동기화)하는 방법으로 NTP(Network Time Protocol)을 이용하여 루트 계정에서 시스템의 시각을 현재 시각으로 설정할 수 있다.

1) SERVER 설정 (Time Server)

① /etc/ntp.conf 구성

```
$ vi /etc/ntp.conf
server 0.0.0.0(맨 마지막 줄 서버IP)
```

② Start xntpdDeamon

```
$ vi /etc/rc.config.d/netdaemons
export NTPDATE_SERVER=0.0.0.0    (ntp server IP Address or hostname)
export XNTPD=1 <-------- 0을 1로 변경
```

③ XNTP Deamon시작

```
$ /sbin/init.d/xntpd start      <-------- Deamon시작
```

④ XNTP 확인

```
$ ntpq -crv
status=c011 sync_alarm, sync_unspec, 1 event, event_restart
system="UNIX/HPUX", leap=11, stratum=16, rootdelay=0.00, rootdispersion=0.00, peer=0,
refid=0.0.0.0, reftime=00000000.00000000 Thu, Feb 7 2036 15:28:16.000, poll=4,
clock=c7bba289.c8740000 Fri, Mar 10 2006 16:00:25.783, phase=0.000, freq=0.00, error=0.00
```

* 여기서 'reftime=' 부분이 0이면 아직 server에서 sync 받지 못한 것임. 이 부분이 16진수로 표시되면 time information을 client에게 줄 준비완료.

```
$ ntpq -crv
status=0544 leap_none, sync_local_proto, 4 events, event_peer/strat_chg
system="UNIX/HPUX", leap=00, stratum=4, rootdelay=0.00, rootdispersion=885.01, peer=2116,
refid=LOCAL(1), reftime=c7bba37a.1c2c2000 Fri, Mar 10 2006 16:04:26.110, poll=6,
clock=c7bba3b1.8ecdb000 Fri, Mar 10 2006 16:05:21.557, phase=0.000, freq=0.00, error=885.01
```

```
$ ntpq -p
   remote           refidst t when poll reach   delay   offset  disp
==============================================================
*LOCAL(1)        LOCAL(1)         3 l   21   64  377    0.00    0.000   10.01
```

* 5분 정도 기다려 이 명령어로 remote부분에 * 표시가 생기면 정상적으로 동작하는 것임.

2) CLIENT 설정

① /etc/ntp.conf에 time server의 IP 설정

```
$ vi /etc/ntp.conf
server 0.0.0.0   {Time Server IP Address 또는 Hostname(/etc/hosts 등록되어 있어야 함)}
```

② 여기에서 주의를 요하는데, clock syncronization 초기화 하는데 있어 ntpdate를 사용하는데 반드시 xntpd daemon이 떠 있으면 않된다.

```
$ ps -ef | grep xntpd
$ /sbin/init.d/xntpd stop
```

* Daemon 떠 있으면 종료한다. 또한 종료되지 않을 경우 Kill 죽인다.(kill -9 사용)

```
$ ntpdate <ip address>
```

③ xntpd시작

```
$ vi /etc/rc.config.d/netdaemons
```

```
---------- 생략 ----------
export NTPDATE_SERVER=0.0.0.0   (ntp server IP or hostname)
export XNTPD=1 (0을 1로변경)
$ /sbin/init.d/xntpd start (Deamon 시작)
```

④ xntpd 확인

```
$ ntpq -p
```

* 5분 정도 기다려 이 command로 remote부분에 * 표시가 생기면 정상적으로 동작하는 것임.

```
$ ntpq -crv
status=0644 leap_none, sync_ntp, 4 events, event_peer/strat_chg
system="UNIX/HPUX", leap=00, stratum=5, rootdelay=0.18, rootdispersion=10.70, peer=46996,
refid=192.168.1.177, reftime=c7bbba2d.7bee7000  Fri, Mar 10 2006 17:41:17.484, poll=6,
clock=c7bbba3b.9a39b000  Fri, Mar 10 2006 17:41:31.602, phase=-0.234, freq=-28.50, error=0.26
```

**server 가 database server 인 경우 시간을 되돌리기는 큰문제를 일으킬 수 있습니다. 따라서 ntp 사용시 time backward 를 disable 하는 기능이 있다. -x option을 사용하여 이 기능을 사용할 수 있다.

```
$ vi /etc/rc.config.d/netdaemons
export NTPDATE_SERVER=ntp server IP or hostname
export XNTPD=1
export XNTPD_ARGS=-x  =>-x option 추가

$ /sbin/init.d/xntpd stop

$ /sbin/init.d/xntpd start
```

* -x option 기능을 사용하면 시간이 되돌려지지는 않고 서서히 clinet쪽 시간을 느리게 하여 시간을 fix 하게 한다.

3) 장애유형

sbin/init.d/xntpd start 명령을 실행하면 xntpd 데몬이 시작되지 않고 다음과 같은 오류가 발생된다.
"socket(AF_INET, SOCK_DGRAM, 0) failed: Too many open files

[해결]
xntpd에 필요한 파일 설명자수는 시스템의 인터페이스 수와 열려 있는 몇 개의 일반 파일에 따라 결정
maxfiles:
==========
xntpd -d -d -d를 실행하면 열려있는 인터페이스 수가 표시된다.
일반 작업의 경우 fd를 10을 더 추가 해야 한다.

기본규칙
maxfiles 60 -> 120 (double it)
나중에 lan 인터페이스를 추가 하는 경우 maxfiles를 늘려야 한다.
nfiles:
========
sar -o temp -v 1 120을 실행하여 nfile이 최대 값에 도달 했는지 확인한다. 그럴 경우 커널 nfile의 크기를 늘린다.
16:54:03 text-szovproc-szovinod-szov file-szov
16:54:04 N/A N/A 103/276 0 0/476 0 355/920 0
                                    ^^^^^^^

추가 maxfiles 또는 nfile을 사용하여 새 커널을 만드는 경우 /sbin/init.d/xntpd start를 실행하여 xntpd 데몬을 시작해야 한다.

4.5 AIX 환경

최근에는 서버/네트워크 장비에 대한 시간 동기화(타임서버 시간 동기화)하는 방법으로 NTP(Network Time Protocol)을 이용하여 루트 계정에서 시스템의 시각을 현재 시각으로 설정할 수 있다.

1) NTP 설명

① NTP(Network Time Protocol)는 UDP 포트 123번을 사용
② 이 포트가 Open되어 잇지 않으면 NTP 서버와 동기화할 수 없음.
③ 8~10분 정도가 지난 후 서버와 클라이언트 간에 시간이 동기화 됨.

2) NTP 서버 구성

① 현재 Timezone / 시간 확인

▶ 현재 Timezone이 어떻게 설정 되었는지 확인한다. 하기 결과창에는 CDT 측, 북아메리카 Timezone으로 설정되어 있다.

```
$ date
Sat Mar 14 01:01:43 CDT 2015
```

▶ 한국에서 일반적으로 "KORST-9" Timezone을 사용하기 때문에 AIX 설치 시 기본적으로 설정되는 "CDT" Timezone을 "KORST-9"로 변경해준 후에 서버 재기동을 해야 한다.

▶ Timezone을 변경하고, 다시 로그인을 하게 되면 Timezone이 KORST로 변경된 것을 확인할 수 있으나, 이는 실제 AIX에 적용된 값이 아닌 변경된 값을 보여 주는 것일 뿐이다. Timezone 변경 후, 반드시 재기동이 필요하다.

```
$ chtz "KORST-9"
```

② NTP Server 설정

▶ /etc/ntp.conf 파일을 하기와 같이 수정한다.
-첫번째로 참조한 Timeserver는 뒤에 prefer를 붙여줌.
-아래 ntp.conf 파일 상에서는 참조한 NTP_Server_IP 뒤에 prefer를 붙여 줬음.
-아래 설정파일을 해석해 보면, NTP_Server_IP를 첫번째로 참조하고, 두번째로 자기 자신의 Local clock을 참조하겠다고 설정한 것이다.

```
$ vi /etc/ntp.conf
#broadcast client
server NTP_server_IP prefer   #NTP Server IP as reference
server 127.127.1.0            #local clock as reference
fudge 127.127.1.0 stratum 0   #values for local clock
driftfile /etc/ntp.drift      #where to keep drift data
tracefile /etc/ntp.trace
```

▶ xntpd daemon 확인

```
$ lssrc -a | grep -i xntpd
```

```
Xntpd      tcpip      inoperative
```

▶ ntp 활성화 정보 확인

```
$ ntpq -nq
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 10.0.0.1        0.0.0.0      3 u    7   64    1    2.884  287.718   0.001
 127.127.1.0     127.127.0.1     16 u    -   64    0    0.000    0.000   0.000
```

③ NTP daemon 시작

동기화 과정에서 NTP Client 측에서 시간이 뒤로 돌아가는 것을 방지하기 위해서, Daemon 시작시, -X option을 준다. (Time backward 방지, 클라이언트 시간 흐름을 조절하여 동기화)

```
$ startsrc -s xntpd -a "-x"
0513-059 The xntpd Subsystem has been started. Subsystem PID is 6946978.
```

④ NTP daemon 확인

▶ xntpd daemon 확인

```
$ lssrc -a | grep -i xntpd
Xntpd      tcpip      inoperative
```

▶ ntp 활성화 정보 확인

```
$ ntpq -nq
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 10.0.0.1        0.0.0.0      3 u    7   64    1    2.884  287.718   0.001
 127.127.1.0     127.127.0.1     16 u    -   64    0    0.000    0.000   0.000
```

3) NTP 클라이언트 구성

① 현재 Timezone / 시간 확인

▶ Timezone은 NTP Server와 동일하게 맞춰 줌.
▶ xntpd는 Server / Client간 1000초(16분) 이상 차이가 나면 더 이상 동기화 하지 않는다.
▶ NTP Server / Client간 시간을 맞추기 위해, Client단에서 #smitty date 명령어를 통해 16분 이상 차이가 나지 않게 설정해 준다. (권장사항은 NTP Server와 가장 근소한 시간으로 맞추는 것)

② NTP Client 설정

```
$ vi /etc/ntp.conf
#broadcast client
server NTP_server_IP prefer   #NTP Server IP as reference

driftfile /etc/ntp.drift          #where to keep drift data
```

```
tracefile /etc/ntp.trace
```

참조하고자 하는 NTP Server IP를 Server 항목에 입력

③ NTP daemon 시작

동기화 과정에서 NTP Client 측에서 시간이 뒤로 돌아가는 것을 방지하기 위해서, Daemon 시작시, -X option을 준다. (Time backward 방지, 클라이언트 시간 흐름을 조절하여 동기화)

```
$ startsrc -s xntpd -a "-x"
0513-059 The xntpd Subsystem has been started. Subsystem PID is 6946978.
```

④ NTP daemon 확인

▶ 대부분의 경우 Reach 값이 377에 다다르면 동기화가 완료된다.
▶ 보통 6~10분 사이에 동기화되며, 바로 시간을 맞추려면 NTP 서버가 active인 상태에서 클라이언트 단에서 "$ ntpdate <ip_of_NTP_Server>" 또는 "setclock <NTP_Server_Hostname>" 명령어를 수행해 주면 된다.
▶ ntpupdate 명령어 수행 후 xntpd daemon을 재기동해 준다.

4) 재기동시에도 NTP 자동실행 설정

① /etc/rc.tcpip 파일 수정

```
start /usr/sbin/xntpd "$src_running" "-x"
```

▶ AIX default 설정 상에는 xntpd이 자동 실행으로 설정이 되어 있지 않음.
▶ /etc/rc.tcpip 파일에서 xntpd와 관련된 라인의 주석을 해제하고 위의 명령어 형태로 수정.

5) 참고사항

xntpd를 이용하여 시간을 동기화 한 후 Time 서버의 시간을 바꾸면 전체 클라이언트의 시간이 바뀐다. Time 서버의 시간을 임시로 바꾸려면 Time 서버 단에서 xntpd를 정지 시킨 후($ stopsrc -s xntpd) 작업한다.

4.6 FreeBSD 환경

최근에는 서버/네트워크 장비에 대한 시간 동기화(타임서버 시간 동기화)하는 방법으로 NTP(Network Time Protocol)을 이용하여 루트 계정에서 시스템의 시각을 현재 시각으로 설정할 수 있다.

NTP를 사용하기 위해서는 기본적으로 NTP 패키지가 반드시 설치되어 있어야 한다. 설치 확인은 다음의 명령어를 실행하여 확인한다. 만약, 설치되어 있지 않으면 "**pkg install ntp**" 명령어로 설치하면 된다.

```
[root]# pkg install ntp
```

ntpd 서비스를 활성화 하기 위해서는 다음 같은 명령어를 사용하여 "/etc/rc.conf"에 등록 해야 한다.

```
[root]# /etc/rc.d/ntpd enabled
```

우리나라에서 운영되고 있는 NTP 서버는 다음과 같다.

```
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net
```

우리나라에서 운영되고 있는 NTP 서버를 ntpd 데몬 설정을 위한 설정 파일인 "/etc/ntp.conf"에 다음과 같이 설정한다.

```
[root]# vi /etc/ntp.conf
#
# NTP
#
server kr.pool.ntp.org
server time.bora.net
server time.kornet.net
```

ntpd 데몬 설정을 위한 설정이 끝나면 반드시 NTP 설정이 제대로 추가되었는지 확인한 후 NTP 데몬의 Restart 작업이 반드시 필요하다.

```
[root]# /etc/rc.d/ntpd restart
ntpd not running? (check /var/run/ntpd.pid).
Starting ntpd.
```

ntpd 시간 확인은 다음과 같은 명령어로 확인할 수 있다.

```
[root]# ntpq -p
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 0.freebsd.pool. .POOL.          16 p    -   64    0    0.000    0.000   0.000
 106.247.248.106 141.223.182.106  2 u    7   64    1    4.412    0.544   0.000
 time.bora.net   204.123.2.5      2 u    7   64    1    5.206    7.741   0.000
*send.mx.cdnetwo 204.123.2.5      2 u    1   64    1    3.968    3.807   0.446
 211.52.209.148  216.239.35.12    2 u    1   64    1   11.862    2.838   0.259
 dadns.cdnetwork 204.123.2.5      2 u    2   64    1    4.833    0.005   0.408
 92.223.73.5 (st 106.247.248.106  3 u    -   64    1    5.015    1.397   0.482
```

* 표시된 ip 가 현재 시간을 가져오고 있는 ntp 서버임

5. About BaroKEY

Version 1.0 - Official Release - 2016.12.1

http://www.nurit.co.kr

상호 : 주식회사 누리아이티
등록번호 : 258-87-00901
대표이사 : 이종일
대표전화 : 010-2771-4076(기술지원, 영업문의)
이메일 : mc529@nurit.co.kr
주소 : 서울시 강서구 공항대로 186, 617호(마곡동, 로뎀타워)